

Security Approaches for Radio Frequency Identification Systems

by

Joseph Timothy Foley

Submitted to the Department of Mechanical Engineering
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy in Mechanical Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

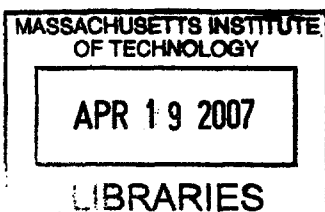
February 2007

© Massachusetts Institute of Technology 2007. All rights reserved.

Author
Department of Mechanical Engineering
February 1, 2007

Certified by
Sanjay Sarma
Associate Professor
Thesis Supervisor

Accepted by
Lallit Anand
Chairman, Department Committee on Graduate Students



ARCHIVES

Security Approaches for Radio Frequency Identification Systems

by

Joseph Timothy Foley

Submitted to the Department of Mechanical Engineering
on February 1, 2007, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy in Mechanical Engineering

Abstract

In this thesis, I explore the challenges related to the security of the Electronic Product Code (EPC) class of Radio Frequency Identification (RFID) tags and associated data. RFID systems can be used to improve supply chain performance and automate asset management. However, an antagonist could use the functionality of the RFID tags and the EPC data to invade personal privacy or acquire access to unauthorized corporate information such as inventory levels. In addition, current RFID mechanisms expose secrets to the readers, which opens an avenue for exploits and information leakage.

I examined the RFID security and privacy issues and designed a number of systems to improve tag authentication, privacy protection, and secure sharing of EPC data. The specific solutions I propose include TagCheck to protect tags from counterfeiting, JanusTag to allow recoverable dynamic recoding of tags, TagFolio for privacy policy enforcement, and TagDirective for secret management and access control. To prevent leakage at the application level during Object Name System (ONS) resolution, I propose the use of an anonymizing TorONS system. Lastly, to protect tags from being “mass killed”, I designed two different categories of RFID tag kill-resistance mechanisms: active protection using Neighborhood Watch communities of readers and Exponential Rampup for tag self-defense. These technologies are combined under one umbrella called TinFoil, creating a comprehensive security solution that successfully protects the data in an EPC-enabled RFID system while minimizing required modifications to existing architecture.

Thesis Supervisor: Sanjay Sarma

Title: Associate Professor

Acknowledgments

- Thesis Advisor and Committee Chair Professor Sanjay Sarma, who provided a great deal of direction, focus, and funding to my research (most of the research in this thesis came out of discussions with him)
- Committee Member Professor Dan Engels, who helped keep the Auto-ID Lab running, for priceless advice
- Committee Member Professor Brian Subirana, whose insight into project management and business were unmatched
- Committee Member Dr. David Brock, who provided many architectural concepts that were critical to the thesis
- Erik Nygren for helping me come up with the original concept of Object Name System (ONS)
- Auto-ID Labs Switzerland for providing a comprehensive survey of RFID technologies
- Steve Weis for peer review and the related research into RFID security
- Roger Dingledine and Nick Mathewson for help interfacing ONS with The Onion Router (Tor)
- Rachel Greenstadt for her debugging of the original TagCheck protocol (previously called KLite)
- Elizabeth Smith for motivating me to finish my research
- my parents, Joe and Mei-Yuk, for giving me the financial support I needed
- my amazing girlfriend, Markéta, for being a wonderful resource, indispensable editor, and necessary distraction

Special Thanks

Alevtina Asarina, Jake Beal, Susan Born, Miriam Boon, Laura Boylan, Martin Feldhofer, Sriram Krishnan, Jenwa Hsung, Simson Garfinkel, Michelle Goldberg, Brynn Kessler, Jennifer Murphy, Dawn Nekorchuk, Benazeer Noorani, Meredith Peck, Sara Pickett, Karl Ramm, Jennifer Selby, Jocelyn “Bink” Tait, Cat Tait, Sanjay’s cat Tigger, Elise Westmeyer, Jim Waldrop, and The Onion King

Biographical Note

Institutions Attended

Massachusetts Institute of Technology	1994-1999
Bachelor of Science in Computer Science and Electrical Engineering	

Massachusetts Institute of Technology	1998-1999
Master of Engineering in Computer Science and Electrical Engineering	

Teaching and Professional Experience

Network Operations Group	MIT	1995-1998
Network Security Group	MIT	1996-1998
2.007 Teaching Assistant	MIT	1999
CSCI E-170 Electronic Security & Privacy	Harvard	2005

Contents

Contents	7
List of Figures	13
List of Tables	15
Listings	17
I Introducing Automatic Identification	19
1.1 Applications of RFID	20
1.1.1 Electronic Locks	20
1.1.2 EZ-pass	20
1.1.3 Money	20
1.1.4 Freight Containers	20
1.1.5 Pallets	21
1.1.6 Electronic Article Surveillance (Anti-Theft)	21
1.1.7 House Arrest	21
1.1.8 Gaming Chips	21
1.1.9 Finding Lost Children	22
1.1.10 Tracking Livestock	22
1.1.11 The Key Questions	23
1.2 Historical Background of RFID	23
1.2.1 The Precursor: Identification Friend or Foe (IFF)	23
1.2.2 Distributed Intelligent Systems Center (DISC)	24
1.2.3 The Auto-ID Center	27
1.2.4 EPCglobal	27
1.3 My Contributions to Automatic Identification	28
1.4 Creating a Secure RFID Solution	30
1.5 Notation	31
1.6 Summary	31

2	Background & Literature Review	33
2.1	Tag Protection	33
2.1.1	Security Measures in EPC Class 1 Generation 2	33
2.1.2	Physical Unclonable Functions	39
2.1.3	Counterfeit Resistance	39
2.1.4	Viruses	40
2.1.5	Private Collision Avoidance	40
2.1.6	Hash Locks	41
2.1.7	Noisy Bloom Filters (NBFs)	43
2.1.8	Blocker Tags	44
2.1.9	Physical Countermeasures	46
2.2	Tag-to-Reader Authentication	47
2.2.1	Elliptic Curve Cryptography	47
2.2.2	Advanced Encryption Standard (AES)	48
2.2.3	Hopper-Blum (HB)	49
2.2.4	Survey of Authentication Mechanisms	51
2.3	Resolution	53
2.3.1	Host Tables	54
2.3.2	Domain Name System (DNS)	55
2.3.3	Hesiod	57
2.3.4	Object Name System (ONS)	60
2.3.5	UCCNet	64
2.3.6	Handle	65
2.3.7	Hybrid Resolution	65
2.3.8	DNSSEC	66
2.4	Privacy Policy Negotiation	67
2.4.1	Platform for Privacy Practices (P3P)	67
2.4.2	UbiComp Privacy	68
2.4.3	Data Guardian	68
2.4.4	Personal Information Carrier (PIC)	68
2.4.5	RFID Acceptor Tag (RAT)	69
2.4.6	Watchdog Tag	69
2.5	Access Control	69
2.5.1	SIREN	70
2.5.2	Enterprise Privacy Authorization Language (EPAL)	70
2.6	Federation	70
2.6.1	Liberty	71
2.6.2	Kerberos	71
2.7	Trust Model	73
2.7.1	Web of Trust	73
2.8	Key Predistribution	75
2.8.1	Resurrecting Ducklings	75

2.8.2	Du-Deng-Han-Varshney (DDHV)	75
2.8.3	Eschenaaur-Gligor (EG)	76
2.9	Anonymization	76
2.9.1	Cricket	77
2.10	Summary	77
3	System Design of a Secure Identification System	79
3.1	Defining Security Itself	79
3.2	Threat Analysis for Identification Systems	81
3.2.1	<i>Ad Hoc</i> Analysis	82
3.2.2	Practical Threat Analysis	83
3.2.3	Case Analysis	84
3.3	Scenarios	84
3.3.1	Military Scenarios	85
3.3.2	Commercial Scenarios	86
3.3.3	Medical Scenarios	88
3.4	Attacks	89
3.4.1	Denial of Service Attacks	89
3.4.2	Man-in-the-Middle (MITM) Attacks	91
3.4.3	Social Attacks	92
3.5	System Design	92
3.5.1	Previous Functional Requirements (FRs)	92
3.5.2	Informational Security FRs	93
3.5.3	Human Limitation FRs	93
3.5.4	Secret Management FRs	95
3.6	Choosing a Key or Hash Size	95
3.6.1	An Issue with EPC Gen2	97
3.7	Lifecycle	97
3.8	Summary	99
4	Tag Identity and Verification	101
4.1	Two-Faced Tags: JanusTag	102
4.2	Introduction to Anti-Counterfeiting	103
4.2.1	Reader-to-Server Assumptions	106
4.3	Hash-Based TagCheck	107
4.3.1	Design	107
4.3.2	Protocol	107
4.3.3	Hash-Based TagCheck Advantages	108
4.4	Hash Functions on RFID	109
4.4.1	Hash Functions Explored by Weis	109
4.4.2	Controlled Physical Random Functions (CPUFs)	110
4.4.3	PGV Hash Function Synthesis	111

4.4.4	Attacks against Hashes	112
4.4.5	Selection of PGV Function	113
4.5	Symmetric Cipher TagCheck Protocol	117
4.5.1	Advantages	117
4.6	Timing Constraints	117
4.6.1	ISO/IEC 18000	118
4.6.2	EPC Gen2	119
4.6.3	Command Interleaving	120
4.7	EPC Gen2 with TagCheck	121
4.8	Summary	122
5	Tag Modification and Transference	127
5.1	TagFolio	128
5.1.1	TagFolio Structures and Requirements	128
5.1.2	Exchange	130
5.1.3	Transport	135
5.1.4	At Home	135
5.1.5	Further Applications and Augmentation	136
5.2	Analysis of TagFolio	138
5.2.1	Cost Analysis	139
5.2.2	Threat Analysis	139
5.3	Summary	140
6	Tag Access Control	143
6.1	The Mass Kill Problem	144
6.1.1	The Kill Protocol	144
6.1.2	Kill Information Management	144
6.2	Protection from Mass Killing	146
6.2.1	Reader Misdirection	146
6.2.2	Listening for “Gunfire” (Neighborhood Watch)	147
6.2.3	Speed Bumps	155
6.3	Protected Kills	160
6.3.1	Proof of Life	160
6.3.2	TagDirective	161
6.4	Summary	164
7	ID Resolution on the Network	165
7.1	ONS Man-in-the-Middle (MITM) Attacks	166
7.2	Silencing/Obscuring the Tags	167
7.3	ONS Anonymization	168
7.3.1	An Anonymizing Proxy: The Onion Router (Tor)	168
7.3.2	TorONS	170
7.4	ONS Misdirection	171

7.4.1	Construction of Fake IDs	172
7.4.2	Using Fake IDs to Detect Attackers	172
7.5	Security Boundaries	174
7.6	Summary	174
8	System View	177
8.1	Tool Review	177
8.1.1	JanusTag	177
8.1.2	TagCheck	178
8.1.3	TagFolio	178
8.1.4	Kill Resistance	178
8.1.5	TagDirective	179
8.1.6	Reader Misdirection	179
8.1.7	TorONS	180
8.2	TinFoil: The Complete System	180
8.2.1	Integration of Individual Elements within the TinFoil Package	180
8.2.2	Tag Operations within TinFoil	182
8.3	Threat Analysis	184
8.3.1	Denial of Service Attacks	184
8.3.2	Man-in-the-Middle (MITM) Attacks	185
8.3.3	Social Attacks	186
8.4	Illustration of TinFoil's Capabilities: A Medical Case Study	187
8.5	Summary	188
A	Related Work	191
A.1	Yellow Pages (YP)	191
A.2	Lightweight Directory Access Protocol (LDAP)	191
A.3	Universal Description, Discovery and Integration (UDDI)	192
A.4	Secure Sockets Layer (SSL)	193
A.5	Zero-Knowledge Proofs (ZKPs)	193
A.6	Leap of Faith	194
A.7	Intuitive RFID User Interface	195
B	Code Distribution	197
B.1	TorLib	197
B.2	Demonstrations and Profiling Clients	197
B.2.1	TorONS.java	197
B.2.2	test-tor-resolve.pl	201
B.3	TorLib	201
B.3.1	TorLib.java	202
B.3.2	TorLookup.java	211
B.3.3	TorResolver.java	229
B.3.4	TorSocketChannel.java	238

B.3.5 TorTCPClient.java	240
Index	245
Bibliography	249

List of Figures

1-1	The Auto-ID EPC architecture	26
1-2	Auto-ID Labs around the world	28
2-1	EPC tag state diagram from [EPC05a]	37
2-2	EPC Gen2 kill flowchart [EPC05a]	38
2-3	Binary tree representation of a 3-bit tag	44
2-4	The OSI and IP model [Ste98]	45
2-5	Example of a DNS resolution path [AL97]	56
2-6	ONS resolution example	63
3-1	The struggle of security	82
3-2	Annotated RFID tag life-cycle	94
3-3	Typical RFID tag lifecycle	98
4-1	Hash-based TagCheck operation	108
4-2	General layout of a synthesized hash function [PGV93]	112
4-3	Block diagram of an AES hash function	115
4-4	Symmetric cipher TagCheck operation	117
5-1	Concept drawing of TagFolio	131
5-2	Pseudo-code representation of a TagFolio tag operation	132
6-1	EPC Gen2 kill flowchart	145
6-2	Flowchart of the Temporary Reprieve kill protection mechanism	149
6-3	Flowchart of the Stay of Execution kill protection mechanism	151
6-4	Flowchart of the Confirmed Kill protection mechanism	153
6-5	Simulation of Exponential Ramp-up kill resistance ($d_0 = 0.001, d = 0.001$)	158
6-6	Proof of Life operation	161
6-7	Diagram illustrating the operation of TagDirective	162
7-1	Step 1: Getting a list of Tor nodes	170
7-2	Step 2: Setting up a virtual circuit	170

8-1	Interfaces where TinFoil elements reduce secret leakage	181
A-1	Ali Baba's strange cave	194

List of Tables

1.1	Typographic notation	32
2.1	EPC RFID classes [EPC05a]	34
2.2	EPC tag header look-up table examples [EPC06]	34
2.3	Comparison of encryption and hash function specifications [FR06] . .	49
2.4	RFID authentication survey, continued in Table 2.5 (reproduced here with permission from [LSMF06])	52
2.5	RFID authentication survey, continued from Table 2.4 (reproduced here with permission from [LSMF06])	53
2.6	Table of DNS server/domain commonality	56
2.7	GID-96 bit divisions	61
2.8	Read result for an example EPC GID-96 tag	61
2.9	Hex encoding result for an example EPC GID-96 tag	62
3.1	Average time estimates for a hardware brute-force attack in 1995 from [Sch96] (y=years, m=months, d=days, h=hours, m=minutes, s=seconds) 96	
4.1	Notation used in hash function analysis	111
4.2	Most secure synthesized hash functions [PGV93]	113
4.3	Gate and transistor counts for AES hash implementation	116
4.4	Link timing parameters [EPC05a]	120
4.5	EPC Gen2 commands with TagCheck [EPC05a]	124
4.6	EPC Gen2 memory map [EPC05a]	125
5.1	TagFolio tag initialization by the store	130
5.2	TagFolio tag state during transport	135
7.1	Benchmarks (in seconds) of <code>tor-resolve</code> and TorONS	171
8.1	TinFoil tag capabilities and size	183
8.2	Tag responses to issued commands inside the TinFoil environment . .	184

Listings

2.1 Hesiod cluster look-up	58
2.2 Hesiod user look-up	59
2.3 Example of a PGP key listing	74
CodePrep/TorONS.java	197
CodePrep/test-tor-resolve.pl	201
CodePrep/TorLib/TorLib.java	202
CodePrep/TorLib/TorLookup.java	211
CodePrep/TorLib/TorResolver.java	229
CodePrep/TorLib/TorSocketChannel.java	238
CodePrep/TorLib/TorTCPClient.java	240

Chapter 1

Introducing Automatic Identification

“One minute. This is the beginning. We’re at ground zero. Maybe you should say a few words, to mark the occasion.”[Uhl96]

What is automatic identification? Shipping operations would claim it is the bar-code system they use to track items on shipping routes. Manufacturing companies often use automatic identification bar-codes in supply chains to gather statistics on production lines. Retailers use various identification technologies for inventories, Point of Sale (POS), and anti-theft Electronic Article Surveillance (EAS) systems. All these systems operate in the same basic way – an object with an identifying number is scanned by a device, then fed into a system that keeps track of its location and other data. It is the need to monitor items that has created the myriad of ID bar-code systems we have today, which are now being enhanced with Radio Frequency Identification (RFID).

RFID has the advantage of longer read range and functionality even when visually obscured, but it does not intrinsically change the functionality of the ID system itself. RFID-enabled systems still read object IDs and help track them. Unfortunately, sometimes these RFID systems are expected to interface with a legacy bar-code system, leading to compatibility complications that make ID interpretation in an open environment challenging. Previous work has created standards to unify many of these applications, including the Electronic Product Code (EPC) architecture, which is designed to standardize identification protocols and encoding while providing customizable resolution and application data services.

The EPC architecture is mainly designed to improve visibility, which was its primary mission. However, widespread use of EPC requires careful consideration of security challenges that may arise. This thesis supplements the EPC architecture by identifying possible weaknesses and proposing solutions to them.

1.1 Applications of RFID

Let us start the examination of RFID technology by considering a few examples of its applications. This will serve to provide context for my work, as well as begin to expose some of the privacy and security challenges.

1.1.1 Electronic Locks

RFID tags are currently used for limiting entry into many buildings, including on the MIT campus. Obvious security concerns, as with any keys, arise if the physical tag is lost. To combat this problem, the Mexican government is requiring its top officials to be implanted with Verichip [Ver] RFID tags [Wei04], small sub-dermal tags marketed as secure invisible house or car keys that cannot be lost. This effort is aimed at improving security at the government buildings and can also help with body identification. However, there are still security concerns with Verichip, as demonstrated in [New06], where Westhues was able to copy, then emulate the reporter's Verichip.

1.1.2 EZ-pass

Automated toll taking is an area that has demonstrated some of the capabilities of RFID to an average citizen [Dix05]. The EZ-pass¹ [E-Z] is a semi-active RFID tag protected by a plastic housing. The tag is read by using back-scatter at EZ-pass-enabled toll stations [Lan05a]. However, even with the convenience that an automated toll taking system provides, concerns about automated speeding ticket issuance and government invasion of privacy causes some people to actively avoid such systems as shown in a recent survey by James *et al.* [Jam05]. Many respondents indicated that they worried about being charged excessive fees and being tracked.

1.1.3 Money

The European Central Bank is investigating embedding RFID tags in their bills to reduce counterfeiting [YY03]. Dutch banks have begun using tags in their cash transfers to ATMs [Col05].

1.1.4 Freight Containers

The concept of putting tracking information on freight containers has existed for a long time, in the form of bar-codes. The freight industry was one of the first to test RFID technology. All railroad containers now have an RFID tag used to track their progress by readers at various transfer points. The tags used in the harsh freight environments are usually active environmentally-hardened tags with access to greater

¹The service and device are called FastLane in Massachusetts, but the technology is identical.

memory, range, and computing capabilities as defined in ISO 10374.2, putting them out of the scope of my research [Lan05a].

1.1.5 Pallets

The seemingly obvious next level of granularity for RFID tracking of goods is in the pallets that items are grouped into. The larger number of tags needed suggests using a cheaper tag. CHEP, a major pallets management firm, initially found that the moisture level in the pallets was too high for their tags. To combat this problem, the current encapsulation method places a tag inside a plastic cover and in the center of the pallet. Even with such success, full adoption has not occurred due to the cost associated with tagging 80 million pallets [Trc05].

1.1.6 Electronic Article Surveillance (Anti-Theft)

In the 1980's, Electronic Article Surveillance (EAS) systems began using modern RFID tags. Sensormatic [Sen] and Checkpoint [Che] have used various EAS technologies including primitive 1-bit RFID tags since their founding in 1960 [Lan05a]. The RF-based EAS commonly uses a tuned coil placed inside a product or attached to it. When the product is legitimately purchased, the coil is disabled or destroyed. The weaknesses of the currently popular EAS systems are further described in Section 2.1.9.3.

1.1.7 House Arrest

RFID tags can be used to ensure compliance with a given rule-set. For example, people convicted of certain crimes and placed on house-arrest can be forced to wear an RFID tag that keeps track of their location. In particular, the sensors in the house tell authorities if the person is still at home [Hou]. The same system can be used as a preventative measure for patients with Alzheimer's disease or a serious mental illness.

1.1.8 Gaming Chips

Gaming Partners International is currently proposing to use RFID to make their gaming chips² harder to counterfeit by embedding tags made by Philips [GPJ04]. In the gaming industry, a counterfeit takes on a more traditional meaning. Chips are equivalent to money inside the casino. Therefore, if an attacker could easily make usable counterfeit tokens, they would be able to rob the casino almost invisibly. Interestingly enough, the Phillips tag, also known as Vegas S, does not contain any

²The term "gambling chips" is not politically correct.

cryptographic features. Security depends upon the unique ID that is programmed and only used as a secondary verification method for the token.

1.1.9 Finding Lost Children

Bluesoft has deployed RFID tracking technology at Legoland Denmark [Col04]. This is an opt-in program that uses 802.11b-enabled active tags made by AeroScout. This application is well suited to its environment because the park is 2.5 million square feet. At a cost of 3 Euros, an AeroScout tag is attached to the wrist of a child and registered to the parent's phone number. The system is able to give a location within 10 feet via triangulation in approximately 20 seconds. The list price of the active tags is \$85, far above what is being considered in this thesis.

1.1.10 Tracking Livestock

Sullivan describes a current program of using RFID livestock tracking to examine the exact path that a cow infected with a disease such as tuberculosis came from [Sul04]. This capability is not new in and of itself, as livestock tagging has been practiced for a long time. The initial tagging in the American colonial era involved branding livestock with a hot iron, or recording the particular coloration pattern of the animal. Since 2001, a number of farmers and ranchers have used RFID systems to keep track of their cattle, and to route the animals to the correct places when they leave their sleeping quarters and go to the fields.

In addition to tracking and routing livestock, RFID technology allows for more up-to-date back-end databases³. The advantage of such a database to collect information associated with livestock became apparent when, in 2004, a cow infected with tuberculosis was found. Within a few hours, the Michigan State Department of Agriculture was able to discern the animal's entire history. Compare this to the two-week time-frame required for a DNA test to confirm that a Canadian cow brought Bovine Spongiform Encephalopathy (BSE)⁴ into Washington state.

Given the potential of up-to-date livestock databases, Sullivan discusses a government proposal for a national database to track all livestock. Tags for such a purpose would cost approximately \$1 [Sul04]. However, despite all the benefits of RFID for livestock tracking, there is resistance to a national registry. As mentioned in [Pec06], cattle ranchers are very public about their brands, but very private about their operations (the equivalent of intellectual property in this industry). As an example, it is considered an extreme faux pas to ask a rancher how many cows or how much land

³The existence of a good database is independent of whether RFID or other technology is used to track the livestock. However, the ease of use of RFID to collect and update the information in the database makes up-to-date databases more prevalent.

⁴better known as Mad Cow Disease

they own⁵.

Before ranchers accept a livestock registry with open arms, their fears that access to such information will not be restricted enough will need to be allayed. Acceptable access could include, for example, animal health officials in response to outbreaks. Ranchers additionally want to see how they can make a profit from such technology and not just be watched by Big Brother.

A practical instance of a previously only imagined concern occurred when meat packers in Canada refused to buy meat from ranches that belonged to a particular industry association because of their affiliation with other groups [Pec06]. If information was always freely available, meat packers could be even more choosy, leading to interesting questions of what could be considered discrimination. The balance between consumer health protection and discrimination is a fine line that needs to be carefully specified.

1.1.11 The Key Questions

Examination of the above examples of RFID applications brings to light the following privacy/security questions to be concerned about.

- Who should RFID tags talk to?
- What are the misuses that are of utmost concern?
- When should the tags talk?
- Who should control verbosity?
- How can they control verbosity?
- How does information and authority get updated?
- How are people and organizations allowed to access the related information?

My work will address all these questions in turn, but before we embark on that journey, it is prudent to consider the historical origin of RFID technology and its subsequent development, particularly at MIT.

1.2 Historical Background of RFID

1.2.1 The Precursor: Identification Friend or Foe (IFF)

The concept of Radio Frequency Identification has existed since the 1940's. The need for such technology grew out of the development of Radio Detection and Ranging

⁵Apparently, the usual response is something like: "How much money do YOU have in the bank?".

(RADAR) in 1935 [RFI05]. RADAR uses high frequency radio emissions to reflect energy off an airplane's profile and determine its location from the reflection. Unfortunately, the reflected information was insufficient to identify the type or intention of the plane. It is believed, for example, that the initial deployment of RADAR detected the incoming planes during the Pearl Harbor attack, but assumed those planes were friendly [Dea05].

The Germans created a system of their own that allowed them to identify friendly airplanes. In response to a special signal from the radar, the plane would roll, causing its signature on the RADAR screen to change⁶ [Dea05]. In 1948, Harry Stockman of the U.S. Air Force Material Command published "Communications by Means of Reflected Power," describing a similar but less crude system for transmitting information via back-scatter [Sto48].

These historical developments set the stage for today's RFID technology. Over the years, Identification Friend or Foe (IFF) systems increased in complexity, but the operating principle remained the same: a signal is sent to the IFF transponder on the plane, and the plane responds back with an appropriate answer that indicates its disposition. In the case of a more sophisticated Fiestel 2-pass system for IFF, the algorithm is still straightforward [RCT06a, FNS75]:

1. IFF interrogators create a random challenge and send it to the aircraft.
2. The friendly plane encrypts the challenge using a shared key and sends the result back.
3. If the interrogator is able to decrypt the challenge, the response is valid.

Fiestel's system was developed in the 1950's and is still used today in a variety of environments, including RFID.

The power constraints and expense on modern military IFF systems are far beyond what can be spent on a given RFID tag for industrial or consumer applications, but the security concerns remain similar. The tighter constraints for commercial applications influenced much of the RFID development at MIT. The research in this thesis focuses on addressing the security concerns of RFID within these constraints.

1.2.2 Distributed Intelligent Systems Center (DISC)

In 1998 at MIT's Laboratory for Manufacturing and Productivity, Dr. Brock and Professor Sarma became interested in using new identification technologies to make automation and robotics easier to control. They founded the MIT Distributed Intelligent Systems Center (DISC), which I became a part of. The center's goal was to make an identification technology that would become ubiquitous to allow automated machinery to make intelligent operational decisions. The additional ID information

⁶Amusingly enough, this is the first recorded usage of RFID back-scatter in an application.

would give greater power to systems like machine vision by providing information to assist in deciphering noisy inputs. Preprocessed data would reduce the computational requirements for information extraction. The technology that quickly became top choice for this application was RFID, in particular for orientation-independent operation on optically obscured items.

Our earliest development was the Electronic Product Code (EPC), which was to have four fields, similar to what is found on a Universal Product Code (UPC):

Version: the version of EPC being read (including the expected size)

Manufacturer: the entity that produced the item

Product: the type of item

Serial: a unique identifying number for that particular manufacturer and product (not found on a UPC)

This information pointed to an entry in a web-enabled database, which was encoded in eXtensible Markup Language (XML) in a schema called Product Markup Language (PML). PML, later renamed to EPC-IS, was developed in collaboration between Dr. David Brock and myself [Bro01]. Determining which database to access was a problem solved with the Object Name System (ONS), an application of the Domain Name System (DNS). The concept of using DNS as a resolution mechanism was conceived during a conversation with Erik Nygren [Nyg99]. The basic architecture of the entire EPC system can be seen in Figure 1-1 with the interfaces between elements highlighted with blue dotted lines. Tags talk to readers, which then talk to applications; applications use ONS to resolve associated servers. These servers are then communicated with via the EPC-IS to execute the desired transactions including retrieval of related information.

I applied the EPC system concepts to our initial prototype of a “smart microwave”, which became the heart of my Master’s thesis [Fol99], completed under the direction of Professor Sanjay Sarma. The operation of the microwave was designed to be intuitive. A user would take their prepackaged and RFID-enabled food item, unwrap it, drop the packaging on top of the microwave, and place the food in the oven. The readout would then give any additional instructions needed⁷ and set the time and power level on the microwave. Upon closing the oven, the microwave was ready to go, and simply required the user to press the START button⁸. This seemed straightforward, but a robust implementation was challenging as will be mentioned in Section 1.3.

⁷such as poking holes in a plastic cover

⁸An initial design had the microwave automatically start when the door was closed, but we decided having the human check things before starting an operation was a good design practice.

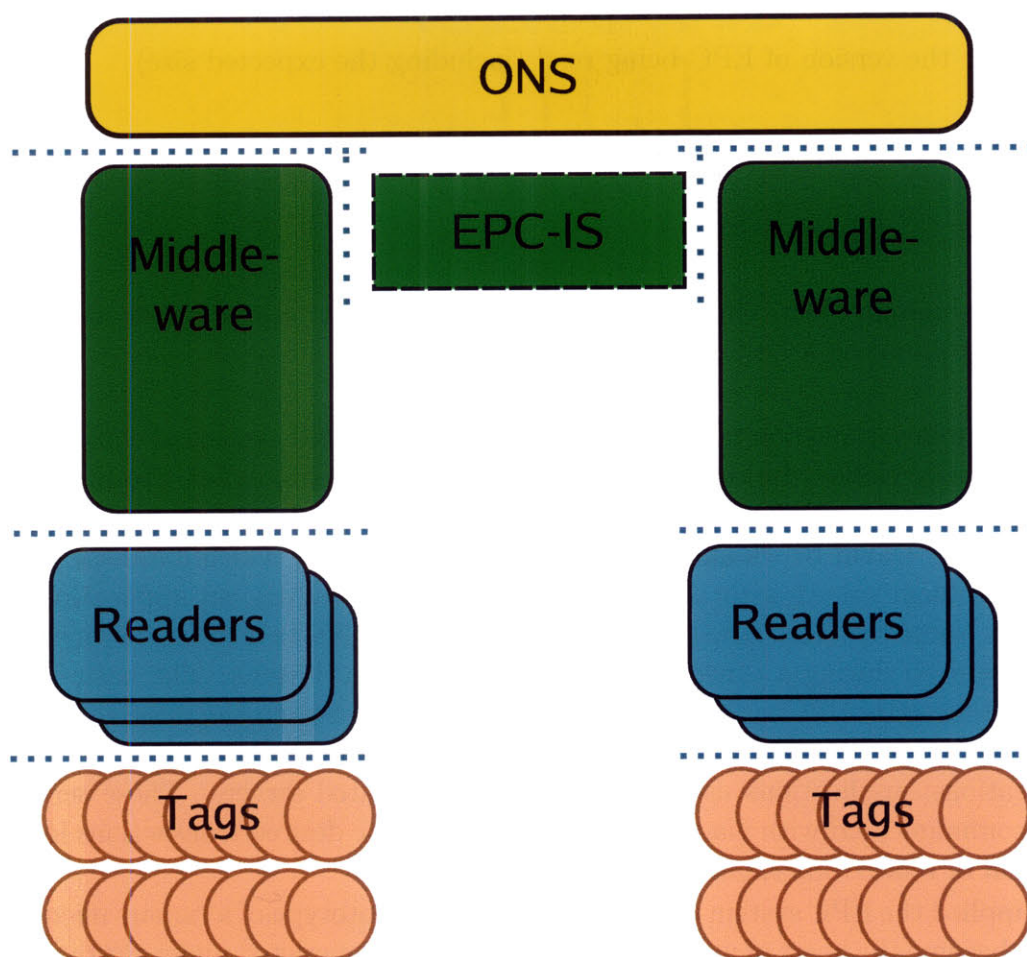


Figure 1-1: The Auto-ID EPC architecture

1.2.3 The Auto-ID Center

On September 30, 1999, The Auto-ID Center subsumed the DISC and took the electronic identifier concept to the next level. While integrating the concepts of ONS and EPC with various other technologies [Sar06], we focused solely on minimalist RFID solutions. A major milestone occurred when this technology was demonstrated to a group of CEOs from a number of industries at Cambridge University in the UK in 2000. The demonstration consisted of a pair of actors interacting with the various EPC and ONS enabled devices that were being developed, including a smart refrigerator and microwave. During this time, Waldrop *et al.* developed the Savant, a minimalist server that administers a small cloud of entities to reduce the deluge of information created by the many events. The Savant processed the information from its entities and answered queries about the information using XML-RPC [XML] or SOAP [GHM⁺03]. In addition, it issued commands to the entities it controlled. The Savant could administer a set of readers, or even a cloud of savants [WE].

Many start-up companies and research projects came out of the Center on various aspects of RFID including RFID-enabled sensors and agile (multiple frequency) readers. The Center itself grew to six labs around the world as shown in Figure 1.2.3.

Eventually, it became obvious that a university based research center was not going to be sufficient to handle the huge industry that was about to explode out of RFID. On November 1, 2003, the Auto-ID Center divided into two entities [Sar06]:

EPCglobal: [EPC] This entity focuses on the corporate and standards aspects under the coordination of the Unicode Council (UCC), now GS1 [GS1]. They analyze industry needs and derive practical standards that fulfill them.

Auto-ID Labs: [Aut] This remains the university research group. Continuing with the same focus as the Auto-ID Center, this group looks at the long-range implications of RFID and investigates concerns that may not have immediate practical implications. They also consider problems from a more academic and theoretical perspective.

1.2.4 EPCglobal

EPCglobal's mission is to manage the various commercial developments related to RFID and the EPC. They work on hardware with vendors and back-end software with development firms. They try to draw up standards that the entire industry can follow for maximum interoperability between the various factions within the commercial sector including manufacturers, resellers, and retailers. They are organized into a number of action groups, each of which focuses on a different element of RFID development and is chaired by a leader in that field.

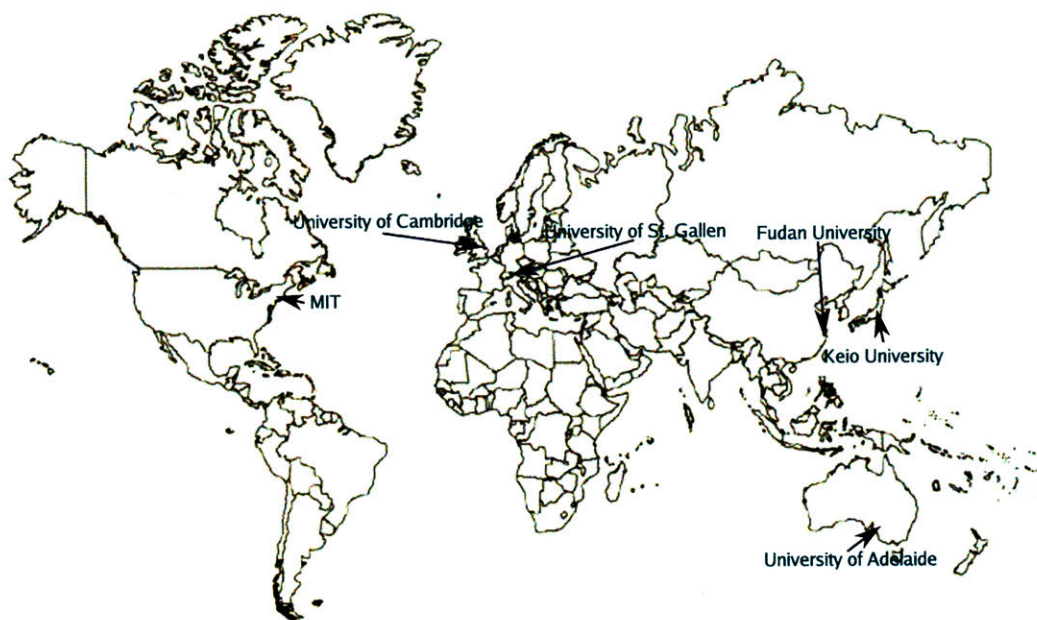


Figure 1-2: Auto-ID Labs around the world

1.3 My Contributions to Automatic Identification

When I began as a graduate student in the Distributed Intelligent Systems Center (DISC), I was given the task of trying to automate a microwave oven using RFID. I built a prototype by connecting an Internet-enabled computer to a micro-controller that interfaced with the buttons on the microwave.

I discovered that the challenge did not lie in figuring out how to interface a micro-controller with the microwave itself, but in creating a system capable of intelligently interpreting the ID that the reader received.

Any developer who interacts with a bar-code, RFID, or any multi-use binary data runs into the problem of interpretation. Given a seemingly random number, determining its disposition is difficult. A given set of bits might point at instructions for cooking a TV dinner but the data could also be a spurious transmission intercepted from the Secret Service [260]. Without structure and a resolution system to indicate what the data is associated with, the number alone is effectively useless.

The traditional prototypical solution relies upon a single database back-end on the application and numbers that all fit within narrow encoding standards for structure and content. I realized that for a useful large-scale system, a single database would be insufficient. Even a massive centralized set of servers would be insufficient for the necessary queries [Fol03a].

Furthermore, the changing needs of various different applications under numerous users would require more than a single encoding specification. Some users might

want an application with only a few types of objects, but a large volume, *e.g.* Mach 3 razor blades. Other products would need many different types of objects, but smaller quantities *e.g.* Boeing 747 parts.

What I therefore needed was a massive decentralized database system. After several attempts at designing one, my colleague Erik Nygren reminded me that such a system already exists [Nyg99] in the form of the ubiquitous Domain Name System. In fact, using Domain Name System (DNS)⁹ as a database had been done with the Hesiod¹⁰ [Dye88] directory system.

I used the previous work of Hesiod to develop a similar architecture, but mapped to the bit name-space of an RFID tag, particularly the 96-bit size specified by the RFID tags available. I called this system the Object Name System (ONS) and it became the backbone for the Electronic Product Code (EPC) [Bro01] architecture. The ONS is a resolution system for mapping IDs to services that uses divisions in the EPC encoding to determine the search path.

Returning back to the microwave problem, the process data on the food item needed a location. To build upon previous work, I created a CGI script on a web-server that would do a look-up into a database and return an XML file containing the relevant product information. The computer that was hooked up to the microwave controller used an XML parser to determine which information was relevant and to act appropriately. This XML language was called the Product Markup Language (PML). Eventually PML was determined to be confusing and replaced with the name Electronic Product Code – Information Services (EPC-IS) [EPC05b].

With the combination of ONS, EPC, and EPC-IS, I was able to complete my automatic microwave project and the protocols and methods used were general enough to be applied to almost any appliance or device. Unfortunately, this capability and convenience did not come without a price. There are a number of scenarios where the device could malfunction in destructive ways in an unexpected interaction with a human. One obvious example was putting the entire item, packaging and all, into the microwave. This would fail to trigger the reader on top of the microwave, and worse, usually cause the packaging to catch fire or melt. There was also a possibility that someone would leave their old packaging on top of the microwave, which would cause the microwave to use incorrect information.

Such problems may seem trivial, but they lead to more interesting questions particularly in the area of security. What if instead of making dinner, a person was being processed for entry into the country (such as the ePassport¹¹ initiative)? In this scenario, any attacks on the system become of great interest, especially to groups like the Department of Homeland Security [CLRPS06]. If the technology is known to be vulnerable to exploits, it becomes of particular value to groups such as illegal aliens and terrorists.

⁹DNS is described in Section 2.3.2.

¹⁰Hesiod is described in Section 2.3.3.

¹¹*a.k.a.* Machine Readable Travel Document

Further concerns exist in the area of privacy [Bro99]. The unique EPC identifier can easily be associated with a particular individual. It may not be the worst thing in the world to have a particular microwavable food item associated with oneself, but consider a person carrying an RFID-tagged medication through an airport. They could now have a particular illness automatically associated with them. Worse still, even though no single individual item may be explicitly aligned with a person, the collection or “constellation” of a number of tags can quickly become an effective predictor of the person’s location [GH06]. This attack even works when groups of non-serialized IDs are used.

For those who worry about government or corporate conspiracies, the tracking of an individual can become alarmingly easy when ubiquitous ID systems are in place. Even disregarding government monitoring, the accessibility of such information may make it possible for arbitrary individuals to invade another person’s privacy or for a corporation to conduct espionage on another corporation.

1.4 Creating a Secure RFID Solution

I believe it is possible to create a next generation secure EPC RFID architecture. The first EPC generation focused on improving visibility. My proposed architecture maintains the same level of visibility while minimizing the leakage of secrets at the communication boundaries between the EPC elements: tags, readers, middle-ware, and resolution services.

There are other systems that have been suggested to try to deal with individual security and privacy concerns, but no end-to-end solution has yet been proposed. The breadth of RFID research requires a general survey of multiple solutions, but only aspects relating directly to an EPC-like architecture will be described in this thesis.

I focus on the design of a minimally augmented EPC RFID system capable of resisting specific attacks. The category of low cost tags that will be introduced are EPC Class 1 Generation 2¹² or of similar capabilities. Only cost-effective augmentations for the analyzed domains will be considered.

This body of research builds upon the implementation details mentioned in other cited research described in Chapter 2 and Appendix A. Analysis of the tag design will show its resistance to current RFID attacks. I will also expose weaknesses that can be explored in future research. We assume that the cryptographic primitives used in this dissertation are secure according to current literature.

The specific proposed solutions span all the major phases of a tag’s life cycle and include:

- Tag identity, validation, and ownership – JanusTag and TagCheck
- Tag death -- Reader Misdirection, and Confirmed Kill

¹²Refer to Section 2.1.1 for EPC tag classification.

- Tag secret management and access control – TagDirective
- Tag modification and transference – TagFolio
- Tag resolution – TorONS

The complete integrated system is called TinFoil.

1.5 Notation

To assist in comprehension of the various domains that are used in this thesis, a number of typographic notations are introduced in Table 1.1.

1.6 Summary

In this chapter, I introduced the problems associated with RFID-enabled applications, particularly in the area of security and privacy. I presented the historical basis of RFID development at MIT to provide context for my work. I summarized my previous work in the RFID arena, which I will build upon to create a next generation secure EPC RFID architecture, spanning the entire life cycle of a tag.

Prior to introducing and analyzing my proposed solutions, however, allow me to review the current literature on the subject in the next chapter.

Typewriter font	programs, URLs, code and related computer output
<i>Italics</i>	formulas and variables
<i>CALLIGRAPHY</i>	computer inputs and outputs, RFID elements
SMALL CAPS	symbolic representations, <i>e.g.</i> WALMART might represent Walmart's Company Code in an EPC
Bold	states and modes
i_j	alternate notation for: j is a subscript of i
<Angle Brackets>	alternate symbolic representation used for clarity or where SMALL CAPS is not available
$[n]$	n bits
$\mathcal{A}$	application
$\mathcal{C}$	computer
$\mathcal{F}$	TagFolio
$\mathcal{H}$	human
$\mathcal{R}$	reader or interrogator
$\mathcal{S}$	server (generic)
$\mathcal{T}$	tag
$\mathcal{V}$	verifier
$H()$	hash function
l_n	hash lock metaID (for locking)
$L_n()$	hash lock n (for unlocking and other operations)
i	random challenge
i	ID for tag or iteration variable
i_p or PID	public ID
i_s or RID	private or secret ID
i	secret (generic)
i_i	secret or password for i
p_i	public key for i
q_i	private key for i
$\oplus$	binary exclusive OR
η	nonce
$ $	concatenation in functions, OR operation in Boolean logic
$E_{\text{key}}(\text{message})$	symmetric cipher encryption
$D_{\text{key}}(\text{message})$	symmetric cipher decryption
$S_{\text{public_key}}(\text{message})$	public key signature

Table 1.1: Typographic notation

Chapter 2

Background & Literature Review

“I know this because Tyler knows this.”[Pal96]

In this chapter, we examine past developments in the field of RFID to gain additional context of where the body of this research lies in current developments.

2.1 Tag Protection

The first step in securing RFID is protecting the tag itself from an outside attack. A number of designs have been developed to address this problem. The ISO/IEC 18000 is an international standard for all RFID tags. I will focus on the Electronic Product Code (EPC) RFID tag systems which are incorporated into that standard as ISO/IEC 18000-6C.

2.1.1 Security Measures in EPC Class 1 Generation 2

The Electronic Product Code (EPC) is an encoding and protocol standard for RFID tags initially developed at the MIT Auto-ID Center¹. The EPC standard sought to provide a low-water-mark for tag manufacturers to follow in the design of their tags such that application systems on top of them would have good inter-operability. The original standard, described in [Bro01], assumed that the EPC would develop around 96-bit tags.

Unfortunately, the multitude of applications and vendor constraints made EPC tag size definition impossible even to this day. Instead, the EPC standard mostly categorizes tags according to their capability as shown in Table 2.1 [GH06]. Occasionally, vendors will desire capabilities that are not in the EPC specification. In this situation, such tags are given a “+” designation, *e.g.* “1+”.

¹previously the Distributed Intelligent Systems Center (DISC)

Class	Definition	Notes
0	read only	programmed by manufacturer (depreciated)
0+	write once, passive	modified Class 0 (not part of EPCglobal specification)
1	write once, passive	programmed permanently by customer
2	rewritable passive	may contain added functionality
3	semi-passive tag	anyone authorized
4	active tag	anyone authorized
5	active tag and reader	

Table 2.1: EPC RFID classes [EPC05a]

Binary	Hex	ID Size	Encoding Scheme
0000 0000	00	N/A	unprogrammed tag
0000 1000	08	64	reserved from depreciated 64-bit standard <SSCC-64>
0010 1111	2F	96	DOD-96
0011 0000	30	96	SGTIN-96
0011 0101	35	96	GID-96
1111 1111	FF	N/A	reserved for future headers longer than 8 bits

Table 2.2: EPC tag header look-up table examples [EPC06]

A separate EPCglobal Tag Data Standard details how various tag standards and sizes are encoded [EPC06]. Some examples of this standard are shown in Table 2.2.

Common terminology used when describing EPC RFID now follows:

Passive tag: A passive tag is powered by the reader and no constant internal power source. Most designs have a small capacitor to temporarily store energy captured from the reader's RF field. The read range on this type of tag is fairly short, depending upon frequency, size of antenna, and sensitivity of the reader. Most tags of this sort use inductive coupling as a power source and transmission mechanism.

Inductive coupling: This method is used primarily by passive tags to transmit information. The antenna coil is switched on and off to modulate the power draw upon the reader's transmission antenna. The changes in power are demodulated into the original data.

Semi-passive tag *a.k.a.* semi-active tag: These tags have an internal power source for powering the controller, but the transceiver uses the reader's RF power to communicate, often in a manner such as back-scatter.

Cover-coding: This method obscures transmission from a tag to a reader. A tag sends the reader a random challenge. The reader XORs the challenge with the data to be returned [EPC05a].

Locking: Locking involves changing the state of a tag such that some operations act differently. This can include not answering certain types of queries, or simply not allowing modification to the data stored on the tag.

EPCglobal [EPC] is the new standards organization that grew out of the initial development at the MIT Auto-ID Center. It is in charge of managing the EPC standard and associated services. In January 2005, EPCglobal came out with its latest specification, known as EPC Class 1 Generation 2, often shortened to “Gen2”. Gen2 tags are expected to encompass the majority of tags that will be used in the retail industry, as they are passive tags (being Class 1), and fairly simple. Gen2 tags are passive-back-scatter, interrogator-talks-first (ITF) tags that operate in the 860-960 MHz range. The reader generates a continuous wave (CW) signal used to power the tag. The tag modulates this signal by changing the reflection coefficient of its antenna, known as back-scatter. The communications are half-duplex, meaning that information is only transmitted in one direction at a given moment [EPC05a]. The tag state diagram is given in Figure 2-1 with the kill-transitions highlighted. In Figure 2-2, we show the Gen2 kill flowchart.

The Gen2 tag has 7 states:

Ready: The holding state of live tags that are not being given an instruction and are not participating in an inventory round². A *Query* command will cause the tag to enter the inventory round by loading a random number into their slot counter and go to the **Arbitrate** state.

Arbitrate: The holding state of live tags who are in an inventory round, but not currently selected for an operation: their slot counter is not zero. When a tag is selected using a *QueryRep* command, it decrements its slot counter.

Reply: Entering the state causes the tag to return a 16 bit random number. When given a valid acknowledgment (ACK) it transitions to the **Acknowledged** state and returns its Protocol-Control, Electronic Product Code and Cyclic Redundancy Check (16-bit version).

Acknowledged: A tag in this state may transition to any state except **Killed**.

Open: A tag with a non-zero access password will enter this state after being authenticated with a cover-coded password and handle. A tag in this state can execute all access commands except *Lock*. **Open** tags may transition to any state except **Acknowledged**.

²Inventory rounds are used in the anti-collision process.

Secured: A tag with a zero access password will transition to **Secured** on a valid *Req_RN*. An **Open** tag with a nonzero access password will transition to **Secured** when that password is given. A tag in this state can execute any access command. A **Secured** tag may transition to any state except **Open** or **Acknowledged**.

Killed: To enter this state, a tag must first be in **Open** or **Secured** state. Tags in these states will transition to **Killed** upon receiving a *Kill* command with a valid non-zero password. After a kill operation is successful, the interrogator will be notified then the tag will not respond to any further queries. A tag once in the **Killed** state will always immediately enter that state whenever powered up.

The relevant EPC feature to enhancing RFID security is the implementation of a “kill password” and “access password” [EPC05a], each of which is 32 bits in size. By default, both passwords are set to zero. While the kill password is zero, the tag cannot be killed. When the kill password is not zero, and a kill command is given, the tag silences itself. The access password is used to put the tag into a secured state. If the password is zero, it is not required for the tag to enter the secured state. The secured state is used to change the various parameters and memory storage on the tag. The operation of the possible states is detailed in Figure 2-2.

Clearly, in this scheme, no complicated cryptography is involved. Instead, access is controlled through simple clear-text passwords blinded by a tag-generated random challenge similar to [MW04]. An attacker who is able to listen to both the tag and reader will be able to execute a replay attack.

2.1.1.1 Medium Attacks

An additional worry with the Gen2 standard is that the KILLED state may not be sufficient to render a tag completely silent. The standard indicates that a tag in the KILLED state still powers up, but fails to respond to any queries [EPC05a]. A mis-implementation or random malfunction of this standard might cause it to resume transmitting later. In addition, even without a reply, information may be retrieved using Differential Power Analysis (DPA) [KJJ99] by examining the dynamics of the tag’s powering up. Another possible attack is Differential Electro Magnetic Analysis (DEMA) [CLRP05], which works by listening to the electromagnetic emanations of a tag via a separate antenna. Even with a properly killed tag, the existence of a coil that can resonate at the proper frequency for RFID might lead to revealing too much information³.

[Jue05b] suggests a set of security modifications for the EPC Gen2 tags. In particular, his solutions make use of an interesting feature: an EPC tag that is sent a kill command but not given enough power to complete the task responds with an error

³Consider how often EAS false alarms occur from various coils people carry.

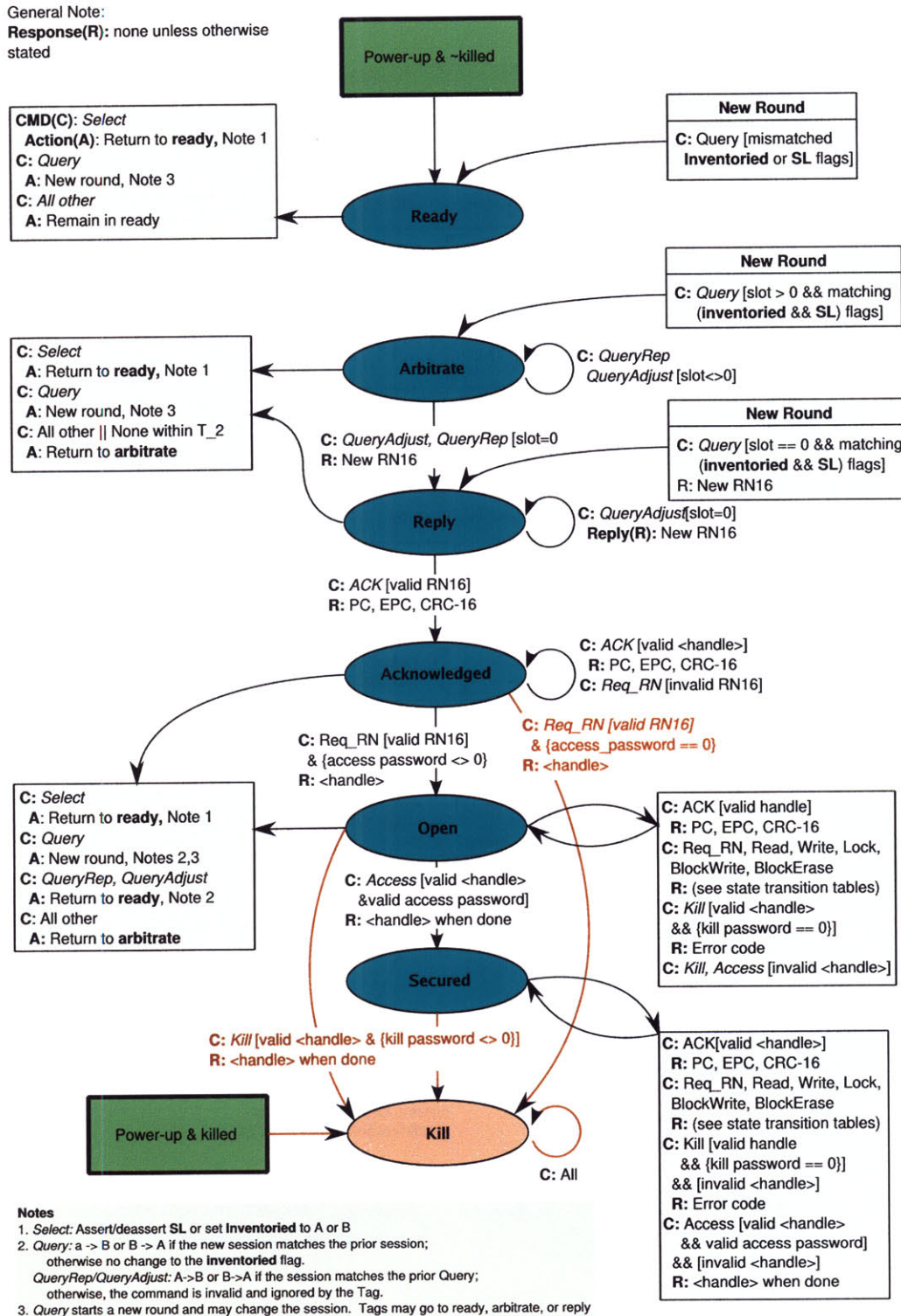


Figure 2-1: EPC tag state diagram from [EPC05a]

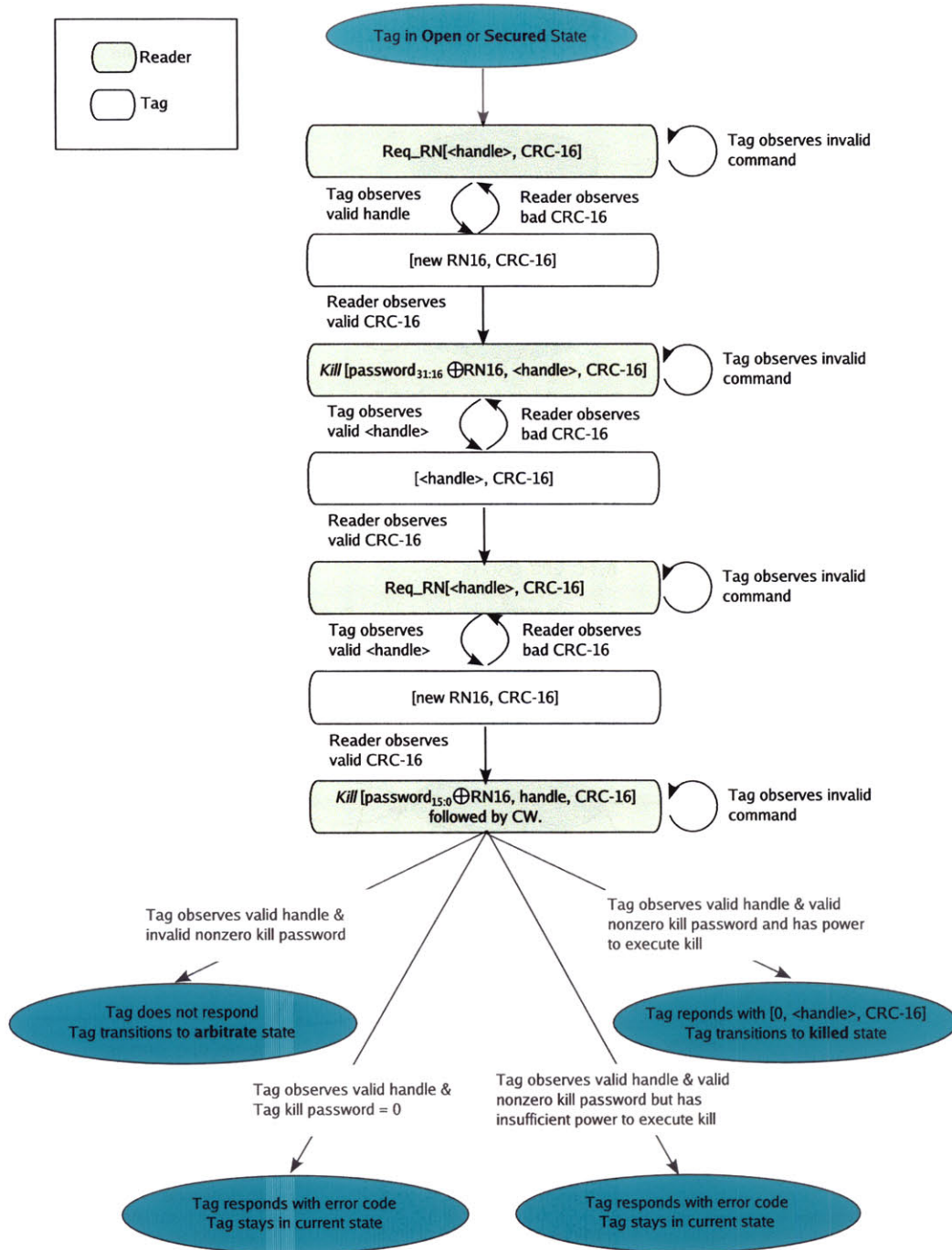


Figure 2-2: EPC Gen2 kill flowchart [EPC05a]

code in the case of a wrong password, but nothing in the case of a correct one. This feature allows Juels to create a side-effect mechanism for verifying passwords without any additional changes to the tag design. This property is used to emulate protocols that require a common secret.

2.1.2 Physical Unclonable Functions

Physical access to the silicon on the tag is another route for an attack. An attacker with an electron microscope and appropriate testing tools may be able to read the individual settings of the transistors on the chip to figure out what the ROM or flash memory contains. As such testing equipment becomes cheaper, the possibility of this type of attack increases. However, there are a variety of tamper-proof mechanisms that exist to reduce the threat. Unfortunately, all of the commercially available mechanisms require an auxiliary system to prevent access to the chip itself or to trigger its self-destruct system. An example of such a system involves the triggering of gadolinium nitrate by an electrical signal⁴, causing the chip to explode [GR02].

Gassend *et al.* have designed another method for resisting microscopic examination, making use of a Silicon Physical Random Function (PUF) [GCvD⁺02b]. A PUF makes use of manufacturing tolerances within a silicon chip to create a unique number unable to be deduced by an outside observer. The number is generated by the differences in lengths of silicon on the chip that are fed through a feedback loop to generate a deterministic set of bits. The wires can be routed to make any analysis extremely difficult without damaging the circuit. Such a circuit is also extremely hard to replicate and characterize because it depends upon manufacturing tolerances.

It should be noted, however, that the difficulty with reproducing the chip's structure does not prevent an attacker from reading such information over the wireless interface to be replayed later, making its application to counterfeit resistance limited. The irreproducibility, which is the technology's strength, also makes it hard to apply to our desired use, as mentioned later in Section 4.4.2.

2.1.3 Counterfeit Resistance

As stated previously, an obvious attack against most RFID tags is to copy the broadcast information, *a.k.a.* skimming. Since a naive tag simply broadcasts its ID to any reader who asks, such an attack is straightforward. It is a common misconception that such attacks are unlikely because readers are complex and expensive. Westhues [Wes06] has proved this to be incorrect by building a reader device for under \$100. His proximity card reader/cloner is a self-contained unit running off 2 CR2032 watch batteries, able to both read and emulate a card it has read. In [New06], Westhues demonstrated the use of the device in a skimming attack to copy the access code of

⁴The chip is coated in gadolinium nitrate during its creation.

a wealthy entrepreneur and spoof the access tag⁵.

Weinstein was able to do the same for the Motorola ASP+ tag embedded into the MIT ID card by reverse-engineering the RFID protocol [MRW04]. He was able to capture and replay multiple cards with equipment costing approximately \$20. The motivation for Weinstein's research was to improve privacy by making it harder for a user to be tracked by his use of the cards.

2.1.4 Viruses

Rieback *et al.* describe a scenario of having viruses in RFID tags [RCT06b]. The attack is not aimed against the tags, but instead focuses on the middle-ware of an EPC system. Using a 128-byte tag, Rieback *et al.* were able to create a self-replicating virus that made use of an SQL injection attack on a simple inventory system. Thankfully, this attack is not as useful for the data sizes proposed by the EPC for tags which are on the order of 128 bits, a factor of 8 too small. However, this does assume that the software coded to post-process the tags does not read more than the given amount of data – if that is not true, then a malicious tag-simulator could simply transmit more bits than the protocol specification allowed. This problem is easily prevented though by the simple solution of using pre-compiled SQL statements that do not evaluate the data as possible instructions. Making use of languages that employ smart memory allocation and type checking like Java makes writing a similar exploit significantly harder.

We should also consider the possibility of an attack that is divided among a number of packets that become virulent when parsed in a particular sequence. Such a situation would be harder to engineer, but is worth considering if the middle-ware is constructed with unsafe structures and evaluation.

2.1.5 Private Collision Avoidance

One legislatively protected area in terms of privacy is that of public library circulation. The FBI under the USA PATRIOT Act is required to have “reasonable grounds” to believe that records sufficiently pertain to an “authorized investigation” [Ass]. Molnar *et al.* addresses the problem of implementing RFID in a library and highlights the possibility of significant invasions of privacy previously believed to be limited to intrusions into the library's database [MW04]. Molnar's inspiration is described in [New06], where he showed that the Oakland Public Library's RFID system was insecure, and even configured so that any individual could rewrite the information.

The main focus of Molnar *et al.* is trying to minimize leakage of information at a very low level [MW04]. The library system generally consists of a single database and meaningless IDs. In some systems, the ID number is simply a serial number of the order in which the books were entered. Such a constrained environment makes it easier

⁵with the owner's previous permission

to build up a useful set of information simply by listening to transactions. This means that the ID should not be visible during anti-collision nor during authentication.

Molnar *et al.* suggest accomplishing this by rewriting the ID each time the tag enters and leaves the library. When a tag is taken to the checkout desk, its number is swapped with a new random but unique one. The mapping is stored in a database for later retrieval when the book is checked back in. This system is the inspiration for JanusTag described in Section 4.1.

Another standard leakage occurs when password protected functions are accessed on a tag. The password is often sent in the clear and reused, making it possible for an attacker to hear the password. Molnar *et al.* propose having a random nonce r generated by the tag that and sent to the reader after receiving a HELLO message. A nonce is a number that is guaranteed to only be used once. This nonce is XOR'd with a secret s , and sent to the tag as an instance password p .

$$p = r \oplus s \quad (2.1)$$

The tag can then extract the nonce out and check the password since

$$s = p \oplus r \quad (2.2)$$

This gives some measure of control to the tag so that it knows what new random ID to listen for when being queried. Such a system does not protect against an active attacker, but it does make the reader-to-tag link much harder to gather useful information from. A physical constraint that improves this type of security is that reader-to-tag transmission power is many orders of magnitude larger than the response from the tag. As a result, it is significantly harder to passively listen to the tag's transmissions. However, the mechanisms are still vulnerable to an active attacker that is listening to both ends.

This example is of particular interest because the EPC Gen2 specification uses the same type of nonce in its cover-coding[EPC05a].

2.1.6 Hash Locks

The idea of one-time passwords/secrets goes back to very early cryptography. Covert operatives in hostile territory were given books of random numbers to use as the key for decrypting messages [Sch96]. The secret used to decrypt a message was a one-time pad, destroyed immediately afterwards. The secrets were often printed on paper designed to be ingested to destroy it. The concept of having a single use password for logging in originated in 1981 with [Lam81]. This inspired Haller to develop S/Key [Hal94], which used cryptographic hash functions to augment the standard UNIX login system. A similar mechanism called a hash lock is described for securing RFID in [Wei03]. The purpose of a hash lock is to provide a method of access to a tag without transmitting the secret. This scheme assumes that a tag is capable of doing

a cryptographically strong hash function and has a writable storage area. Tags start unlocked, which allows any reader to lock them. To lock a tag with the ID i using this method:

1. The configuration agent creates a key L .
2. The configuration agent calculates a metaID l with the calculation $l = H(L)$.
3. The configuration agent sends l to an unlocked tag and tells it to lock.
4. The configuration agent stores the l and L pairing for later retrieval.

While a tag is locked, it will only respond with its l . To unlock the tag:

1. The application reads the l from the tag.
2. The application looks up the L for that l .
3. The application sends the L to the tag.
4. The tag compares $H(L)$ to its l' to determine if it is a valid unlock request.
5. If the two match, the tag is unlocked and will reveal its identity.

The above scheme would allow the reuse of L for multiple accesses. Unfortunately, doing so introduces insecurity back into the system. An attacker that sees L go by can now access other tags. The correct intermediate solution is to apply a policy to always leave tags locked. As soon as an application is done with its modifications in a safe environment, the tag should be locked. With the initial state of all tags unlocked, a malicious attacker could simply write random numbers as l to the tags to keep anyone from accessing the real l . Such issues will be addressed in Chapter 6, where I discuss secret sharing.

Fitting a hash function on a low-cost RFID tag is not a solved and simple task [Wei03]. With one of the implementations or the synthesized hash function that I describe in Section 4.4.3, we will have a working hash function.

However, hash locks do not solve the problem of tracking. An augmented version of the protocol called a “randomized hash lock” which requires a PRNG was developed for tracking resistance. The protocol includes the following steps:

1. The reader sends a query.
2. The tag generates a random nonce R .
3. The tag creates $H(ID||R)$.
4. The tag returns $R, H(ID||R)$

The problem with this protocol is that the reader needs to know what IDs it should be looking for. As the number of possible IDs grows, (as is the general sparse case), this solution becomes impractical. This limitation will prove to be a common one among the systems that prevent tracking. The suggested solution is to seed the database via a second channel such as bar codes or printed human-readable information. However, this is problematic as it exposes the ID we're attempting to protect to anyone with physical access. On the positive side, the tag no longer needs to be unlocked in this scenario, as the reader knows what its ID. Other anti-tracking methods are described in the next subsection.

2.1.7 Noisy Bloom Filters (NBFs)

Another idea for improving security of a tag mentioned in [DWIO05] is to use a noisy Bloom filter. A Bloom filter is a set of hash functions that encode a data set via a different bit-based representation. To calculate the filter's output, the input x is run through k independent hash functions $h_k(x)$ with output values in the range of 1 through m . The value of the $h_k(x)$ bit in the output is set to 1 for each hash. These k bits, equivalent to the original ID, are concatenated together to effectively make a new composite hash function.

A noisy Bloom filter (NBF) simply takes the output of the Bloom filter and applies a noise imposer on it. The imposer examines the 0's in the hash and changes 0 into 1 randomly until a saturation value s is reached. The saturation value creates an artificial signal to noise ratio, which obscures the valid values. s is typically 0.5.

In order to verify if a given tag was created by a noisy Bloom filter, we apply the same Bloom filter and examine all bits that are set to 1. If any bits in the Bloom filter that are set to 1 correlate to a 0 on the NBF, the filter it is guaranteed to not be the correct one. Otherwise, the likelihood of a match is high. The false positive rate is estimated to be s^k .

An obvious attack on this system is to repetitively query a tag, looking for bits that change. A real 1 will always stay a 1. A real 0 will be seen in the flipping values over the course of a number of queries. This mechanism is unable to use the "learning parity with noise" advantage that the Hopper-Blum (HB) series of protocols discussed in Section 2.2.3 uses because the multiple hash functions are deterministic, unlike HB's random inner dot product challenges. NBF also suffers from the need to know which tags it will read ahead of time.

Further development of the NBF system is described in Doi *et al.*'s Kallima [DWI⁺05], a system for protecting the tag-reader interface while minimizing tag modification. The tag proposed here has a precomputed Bloom filter (or set of filters) in read-only memory. To perform NBF, a noise imposer is applied to the Bloom filters before transmission. To combat the problems that a constantly changing ID causes for anti-collision algorithms, Doi *et al.* propose having the random number generator in the tag produce a meta ID that will be used only for anti-collision and stored in

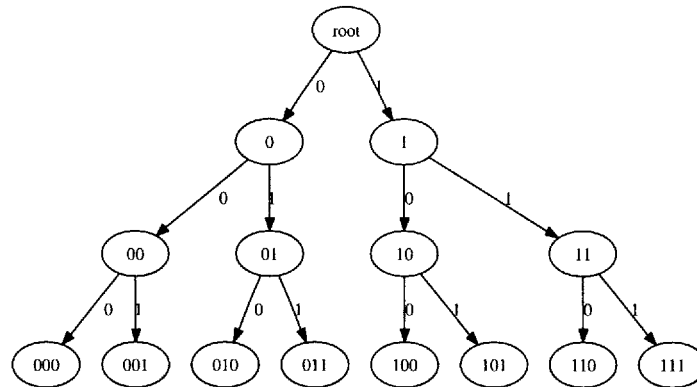


Figure 2-3: Binary tree representation of a 3-bit tag

SRAM for a particular time-period before being generated again.

2.1.8 Blocker Tags

RSA has developed an interesting method for protecting RFID tags. They describe a “blocker tag” that prevents other tags from being read [JRS03]. The method blocker tags use to operate is quite elegant exploiting a property of the binary tree-walking system of anti-collision.

The binary “tree-walking” anti-collision algorithm is quite simple. The reader sends out a general query and listens for any collisions from the tags. If there are collisions, the reader begins the anti-collision algorithm. Each tag’s series of bits represents a binary tree, with a bifurcation at each bit as shown in Figure 2-3.

A “tree-walking” algorithm completes depth-first search of the tree-space by asking tags incrementally what their value at a given index is. When there are collisions of a 1 and 0, the algorithm records a branch to be explored later and tells one of the branches to be quiet. It then explores the active branch, continuing its depth-first search. Once all the collisions are resolved, all present tags are accounted for.

This algorithm is deterministic and efficient. Unfortunately, it does require the reader to keep track of significant amounts of information and possess the ability to temporarily silence tags. Furthermore, using this method allows a passive attacker to see what tags a reader is accessing simply by watching the requests the reader makes while querying the tags. However, this weakness can also be exploited to improve privacy as shown in the development of the blocker tag. In its simplest form, the blocker tag device answers yes to any query about the presence of a 0 or 1 at all bits. This effectively emulates all tags in existence, hiding the presence or absence of any other tag.

Blocker tags have a great potential for improving personal privacy with respect to RFID tags, both in terms of tracking and associated information. Unfortunately they

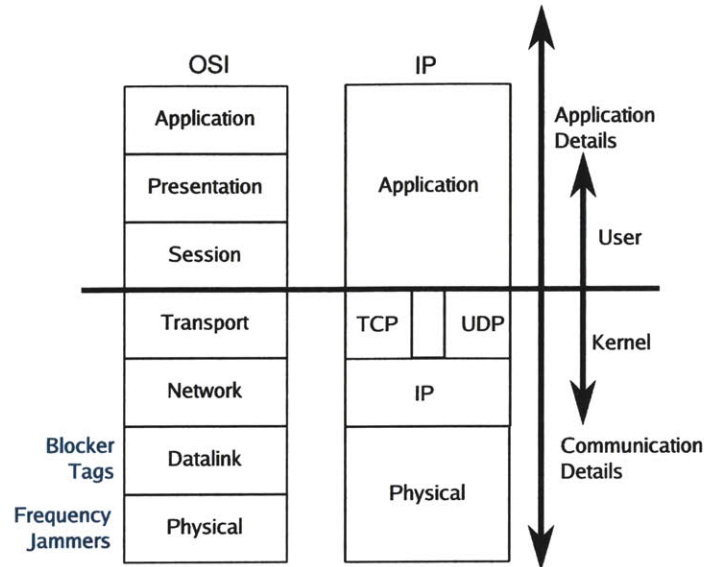


Figure 2-4: The OSI and IP model [Ste98]

suffer from two significant weaknesses. First, they only work on tags that specifically use the binary tree walking anti-collision algorithm. Second, the potential for abuse is high.

The second weakness may be the reason that blocker tags have not become widespread. For example, to cause major problems in a store inventory or anti-theft system, an attacker simply needs to activate a number of blockers, throwing the shelves into disarray as all the various tags disappear in the noise. If the inventory system were written naively, such an attack might even corrupt the location database with all the imagined tags.

A slightly different system functioning along the same lines is a selective blocker. This blocker would only mask parts of the ID and warn any reader that the rest was being forged. Unfortunately, this has similar potential for abuse. The selective blockers would still cause an inventory system to become desynchronized if any of the IDs disappeared. If serial numbers were blocked, the inventory system would not be able to keep track of a large number of units disappearing.

Such potential abuses need to be kept in mind as we consider a protective RFID system. Commercial success may make security enhancements ubiquitous. An interesting consideration is that these solutions are simply jamming at different architectural levels. A spark-gap generator or a Tesla coil is able to jam a variety of RFID by overwhelming all local valid sources. Considering this in the context of the Open Systems Interconnect (OSI) model (Figure 2-4) [Ste98], these radio frequency jammers work at the physical layer (layer 1). A blocker tag does the same thing, but at a higher level in the protocol architecture, at the data link layer (layer 2).

As an alternate protection, Juels *et al.* propose a cooperative effort to develop a “soft blocker” [JB04]. A soft blocker scheme makes use of a Tag Privacy Agent (TaPA) to enforce a security policy that the user desires. If a reader tries to read a field where tags are present, TaPA checks to see if a blocker tag is present. If it is, TaPA only releases public information. If there is no blocker within TaPA’s read range, it returns both public and private information.

An optimal solution would expose only the data that a user wished it to expose. It would act similarly to a firewall in the case of network security – filtering out what should not get through, but exposing information as required.

2.1.9 Physical Countermeasures

In addition to the highly technical methods for protecting tags from being read introduced thus far, let us consider a few of the old-fashioned ones as well.

2.1.9.1 Avoidance

Avoidance is seemingly simple, we could just not allow RFID tags anywhere near us. Unfortunately, friendly government regulations allowing us such a choice and a method for finding all tags would be necessary for a successful implementation of avoidance. As RFID technology becomes ubiquitous, this lifestyle choice proves practical.

2.1.9.2 Destruction

How about physically destroying any tags we come across? This is similar to how some anti-theft systems disable their mechanisms. The audio-resonance tags in DVDs and CDs are deformed so they cannot resonate at the same frequency. For RFID tags, overloading the tag with RF is usually sufficient to achieve a similar effect. If you don’t mind some smoke and minor damage to the product, a microwave also works quite well⁶. However, such techniques are problematic if the tag is needed later or if it is enclosed in a protective casing designed to prevent its death.

2.1.9.3 Barrier

A physical barrier that is impervious to energizing RF⁷ and/or transmitting signals⁸ could be placed around a tag keep it from talking to a reader. When you purchase an EZ-pass [E-Z] transponder, you also receive a Mylar-lined bag. This bag is sufficient

⁶During initial testing of the Motorola Bistatix tags, we decided to see what microwaving did to them. The silicon on the tag exploded with a very large pop, while the rest of the tag caught flame and burned to ash in seconds.

⁷for passive RF tags

⁸for all RF tags

to keep any RF that the tag might transmit from reaching the reader at the toll booth. With the use of the bag, the user has control over when the tag is available to be read and when it should be silent. Needless to say, the practice of bagging and unbagging tags is less useful when tags begin appearing in all products including the ones a person wears. The only barrier option that might work in such a situation is a full metal-shielded bunny-suit.

The idea of using barriers to silence anti-theft Electronic Article Surveillance (EAS) devices is already commonly used in the form of metal-lined insulated bags known as “booster bags”⁹. These bags are usually sold in stores as a way to keep frozen products cold longer by minimizing heat transfer. The thick insulation prevents diffusion and convection, while the metal lining reflects irradiated heat. Both types of insulation also act as an effective sound insulation (against ultrasonic systems) and a Faraday cage (to prevent transmission of electromagnetic waves as in RFID tags). These bags have been re-purposed to make EAS-tagged items unable to communicate with readers at the gateway points [GH06].

In the future, it may become common practice or even good marketing to start use booster bags to protect one’s privacy in an RFID-enabled world.

2.1.9.4 Certification

Technologies to prevent copying and tampering have existed for a long time. Hundreds of years ago, messages were sealed with wax imprinted with a unique mark used for authentication. Paper money incorporates a variety of special features making it difficult to copy including microfilm, micro-printing, and water marks. Credit cards now use special materials in the signature are making attempts at modification obvious. Each of these technologies works well in the physical world, but they do not translate well to RFID. Cryptographic signatures require significant processing power in order to make use of hashes and public key cryptography.

2.2 Tag-to-Reader Authentication

In order to secure the link between a tag and a reader, we need some form of cryptography. Without cryptographic protection, an eavesdropper will have access to all communications between the devices.

2.2.1 Elliptic Curve Cryptography

An area of cryptography that may have promise for RFID public key encryption is elliptic curve cryptography, first proposed by Mille [Mil86] and Koblitz [Kob97] in the

⁹“Boosting” is a slang term for theft.

mid-eighties [Sec04]. The operation of such systems is very similar to currently public-key cryptosystems except that modular arithmetic is substituted with operations defined over elliptic curves. The elliptic curve discrete logarithm problem can be described as follows:

Given two points G and Y on an elliptic curve such that $Y = kG$ (Y is G added to itself k times.), find the integer k . [Sec04]

Current algorithms for computing general elliptic curve discrete logarithms are inefficient, allowing for small key sizes, which in turn allows smaller key size allows hardware implementation to be more compact. Two implementations, Kumar *et al.* [KP06] and Batina *et al.* [BGK⁺06], have brought the number of gates to the order of 10,000-20,000. Unfortunately, even these implementations are still too large for the minimal cost RFID tag.

2.2.2 Advanced Encryption Standard (AES)

The government standard for symmetric key cryptography until very recently was the Digital Encryption Standard (DES). Unfortunately, with its very small key size (56 bits), it was eventually deemed insufficient. In 2001, the National Institute of Standards and Technology (NIST) announced adoption of the Advanced Encryption Standard (AES), which makes use of the cipher Rijndael [FS03a].

Previous implementations of various symmetric ciphers focused on speed instead or size, resulting in sizes on the order of 50,000-100,000 gates. The advent of RFID has refocused such research into minimizing gate counts, detailed in Table 2.3. Poschmann *et al.* refined an implementation of Digital Encryption Standard (DES) called DESL that fits in 1,848 gates [PLSP06]. Feldhofer *et al.* describes new methods of manufacture and a redesign of AES such that it fits on 0.25mm² of silicon [FWR05], with a size of approximately 3,400 gates (10,200 transistors) and bandwidth of 9Mb/s at 125kHz. With this significantly smaller implementation available, Feldhofer proposes simple RFID challenge-response mechanisms for authentication that rely on symmetric cryptography including AES [FDW04, Fel04].

The miniaturized AES will be a keystone to many of the protocols I designed as part of this thesis including generation of a hash function and pseudo-random number generator. AES with respect to these protocols is detailed in Section 4.3.

A similar but slightly different approach to minimizing AES is described in [KN05]. Instead of using AES directly, the proposed design employs a simplified matrix arithmetic encryption mechanism that depends upon the difficulty of recovering the multiplicand or multiplier from the product of matrix multiplication. A liability of this protocol has yet to be verified and no implementation specifications were given in the paper.

Algorithm	Gate Equivalents	I_{mean} 100kHz	Clock Cycles
SHA-256 [FR06]	10,868	15.87	1,128
SHA-1 [FR06]	8,120	-	1,274
MD5 [FR06]	8,400	-	612
MD4 [FR06]	7,350	-	456
AES [FR06]	3,400	8.15	1,032
DESL [PLSP06]	2,309	1.19	144

Table 2.3: Comparison of encryption and hash function specifications [FR06]

2.2.3 Hopper-Blum (HB)

A very interesting authentication mechanism proposed in [JW05] is based on a human-to-computer authentication mechanism called Hopper-Blum (HB) originally described in [HB01]. Hopper and Blum noticed that most human to computer authentication mechanisms required additional hardware or some measure of trust in the communication channel. They developed mechanisms that meet this need in the face of both a passive and active attacker with an insecure communication channel. The key to the protocols is a provably NP-hard problem called “learning parity in the presence of noise,” described below¹⁰.

A binary inner product is a two-step process for generating a “parity” bit. Two m -bit vectors are compared bit-by-bit by the AND operation. If, at a given position, both bits are 1, a temporary m -bit vector has its bit set to 1. If not, the bit is set to 0. Once all bits have been compared, the binary sum of the temporary vector’s bits is performed. If the sum is even, the product is 0. If the sum is odd, the product is 1.

A Galois field, $GF(p^i)$, is a finite number field with p^i elements in it, where p is prime, and $i \geq 1$. The Galois Field used here is $GF(2)$. $GF(2)$ can be represented simply by 0 and 1, which maps well to a computer’s bit representation. I will therefore be using the term “bit” and $GF(2)$ interchangeably.

Prior to using the Hopper-Bloom mechanism, the computer $\mathcal{C}$ and human $\mathcal{H}$ share a secret. The secret is a n -bit vector $\mathbf{x}$. The actual Hopper-Bloom then operates in the following way:

1. $\mathcal{C}$ generates $\mathbf{c}$, a random n -bit vector “Challenge”.
2. $\mathcal{H}$ responds with the binary inner product: $r = \mathbf{c} \cdot \mathbf{x}$.

¹⁰NP stands for “Non-deterministic Polynomial time” indicating such problems to be solvable in polynomial time on a non-deterministic Turing machine. More intrinsically, NP problems can be verified in polynomial time. NP-hard refers to problems that are at least as hard as any decision problem in the NP. More informally, NP-hard problems are problems that require significant computing power to solve in the general case [CLRS01].

3. The sequence is repeated k times.

The probability that a valid user will succeed on any given iteration is 1, while an attacker will succeed $\frac{1}{2}$ of the time. This means that for a given session, the probability of successful attack becomes $\frac{1}{2^k}$. This is very similar to the argument used in a traditional zero knowledge proof [QG90]. A limitation of such an implementation is that an adversary can use Gaussian elimination to determine $\mathbf{x}$, after observing $O(n)$ iterations.

To make this system significantly harder to attack, Hopper and Blum introduce the noise parameter $\eta \in (0, \frac{1}{2})$. We then allow $\mathcal{H}$ to respond incorrectly with probability η . The presence of the incorrect responses makes Gaussian elimination impossible as long as there is no leakage about what responses were valid.

Even with the introduction of noise into the system, it is still vulnerable to an active attacker. If an attacker is able to query the human and keep $\mathbf{c}$ the same each time, or systematically change the values to stabilize values, it quickly becomes clear which values are noise, and Gaussian elimination can be used again.

Juels *et al.* take the HB protocol one step further in HB+ by having $\mathcal{C}$ initially authenticate to $\mathcal{H}$ in order to create the $\mathbf{c}$ [JW05]. In effect, the reader and the tag build the challenge simultaneously, making an active attack much harder. Since we are now talking about tag and reader, we will be substituting $\mathcal{T}$ for $\mathcal{H}$, and $\mathcal{R}$ for $\mathcal{C}$ in the original protocol. Both $\mathcal{T}$ and $\mathcal{R}$ have shared secrets: k -bit secret $\mathbf{x}$ and k -bit secret $\mathbf{y}$. This modification makes the protocol secure even in the face of an active attacker as proven by Katz *et al.* [KS06].

A single round of the protocol consists of these steps:

1. $\mathcal{T}$ generates a random k -bit “blinding” vector $\mathbf{b}$.
2. $\mathcal{R}$ challenges the tag with a k -bit challenge $\mathbf{a}$.
3. $\mathcal{T}$ computes $z = (\mathbf{a} \cdot \mathbf{x}) \oplus (\mathbf{b} \cdot \mathbf{y}) \oplus \eta$ and sends it to $\mathcal{R}$.
4. $\mathcal{R}$ accepts the round if $z = (\mathbf{a} \cdot \mathbf{x}) \oplus (\mathbf{b} \cdot \mathbf{y})'$.

The tag is authenticated if the tag’s response is correct more than our set threshold of times.

The HB+ protocol has the advantage of a very low transistor count for implementation: on the order of a few hundred. However, it takes many interchanges between the reader and tag to get to an acceptable authentication which may not be acceptable in a high-latency network. To make matters worse, it is not an absolutely deterministic protocol because of the allowed errors. To make this protocol useful, a certain amount of tuning is required to set η is set to a reasonable value such that the transactions are protected but the amount of wasted bandwidth is minimized.

The HB+ protocol will not be considered for our application because it requires the reader to know the secret in order to authenticate. Since readers are the first

avenue of attack, this leads to a very vulnerable system. We need a system that never places the shared secret within reach of an attacker. The current protocol also requires that a tag expose a consistent identity which can be used for unauthorized tracking. We will however propose using HB+ in our final design as a separate authentication mechanism.

2.2.4 Survey of Authentication Mechanisms

Lehtonen *et al.* completed a survey of many current RFID-targeted authentication technologies [LSMF06]. I have included the survey's table in the interest of brevity, in Table 2.4 and Table 2.5. Some previous work is covered in greater detail in the previous sections.

Approach	Tag Memory		Tag Function				Network Req				Reader Req	
	NVM	Read-Write	Bit-wise Operations	PRNG	Hash Function	Symmetric	Public	Semi-Public	Secret	Complex DB	Complex Reader	Physical Verification
Unique Serial [STF05], [KSCB03], [Pea05]								•				
Track and Trace [STF05], [KSCB03], [Pea05]								•		•		
Juels [Jue05b]									•			
Ranasignhe [REC04] ^a									•			
Pearson [Pea05]	•	•					•				•	
Juels <i>et al.</i> [JP03]	•	•					•				•	• ^b
Zhang <i>et al.</i> [ZK05]	•	•					•			•	•	
Juels [Jue04]	•	•	•						•			
Vajda <i>et al.</i> [VB03]	•	•	•						•		•	
Tuylset <i>et al.</i> [TB06]	•		•								•	
Juels <i>et al.</i> [JW05]	•		•	• ^c					•		•	
Piramuthu [Pir06]	•		•	• ^c					•		•	
Tsudik [Tsu06]	•	•	•	•					•	•		
Chatmon <i>et al.</i> [CvLB06]	•	•	•	•					•	•		
Duc [DPLK06]	•	•	•	•					•	•		
Molnar <i>et al.</i> [MSW05]	•		•	•					•		•	
Engberg <i>et al.</i> [EHDJ04]	•		•		•				•		•	
Avoine <i>et al.</i> [AO05]	•	•	•		•				•	•		
Gao <i>et al.</i> [GXW ⁺ 04]	•	•	•	•	•				•	•	•	
Rhee <i>et al.</i> [RKKW05]			•	•	•				•	•		
Dimitrou [Dim05]	•	•	•		•				•	•		
Yang <i>et al.</i> [YPL ⁺ 05]	•	•	•		•				•	•		
Weis <i>et al.</i> [WSRE03]	•		•		•				•		•	

^a needs a physical unclonable function (PUF)

^b optional

^c only one random bit required

Table 2.4: RFID authentication survey, continued in Table 2.5 (reproduced here with permission from [LSMF06])

Approach	Tag Memory		Tag Function				Network Req		Reader Req	
	NVM	Read-Write	Bit-wise Operations	PRNG	Hash Function	Symmetric	Public	Semi-Public	Secret	Complex DB
Henrici <i>et al.</i> [HM04]	•	•	•		•				•	•
Lee <i>et al.</i> [LAK06]		•	•	•	•				•	•
Choi <i>et al.</i> [CLL05]	•	•	•		•				•	•
Lee <i>et al.</i> [LHLL05]	•	•	•		•				•	•
Molnar <i>et al.</i> [MSW05]	•	•	•	•		•			•	•
Dimitrou [Dim06]	•		•	•		•			•	•
Felhofer [Fel03] [Fel04]	•		•			•			•	
Bailey <i>et al.</i> [BJ06]	•		•	•		•			•	
Dominikus <i>et al.</i> [DO05]	•		•	• ^d		•			•	
Nochta <i>et al.</i> [NSF06]	•		•				• ^d			

^d not necessary for all proposed approaches

Table 2.5: RFID authentication survey, continued from Table 2.4 (reproduced here with permission from [LSMF06])

2.3 Resolution

Many simple RFID systems assume that the resolution of ID to application has already been done. The ID's domain and access control is pre-determined, making it clear what operations are to be applied. This is sufficient for simple closed-environment use, but in a real environment, such an assumption can be dangerous. Modifications to or incorrect identification of an electronic passport as a stolen item could have serious consequences. Resolution is necessary to identify which specific system is able to cope with a given ID. This section discusses the history of network resolution services and access control developments that have allowed my research to progress.

2.3.1 Host Tables

In the 1960's, when the Internet was still known as the ARPAnet¹¹, the number of hosts on the network was fairly small (on the order of a few hundred machines world-wide) [AL97]. The ARPAnet was developed to allow researchers at one development center to use computing resources elsewhere. The concept of a distributed network without a central controller ensured that the network would continue to function in the face of a massive nuclear attack. The capabilities of this network were primarily limited to serial terminal control of the remote machine [LCC⁺00].

TCP/IP (Transmission Control Protocol/Internet Protocol) was developed in the early 1980's and became the standard host to host protocol with a new addressing scheme. The simplicity of TCP/IP and its inclusion in BSD UNIX led to university involvement, which brought the number of machines to 60,000 and the name Internet was coined.

In order to use the Internet, the user needed to know the address of any machines they were trying to access, which looked something like "4"¹² or later "18.18.0.249"¹³. Keeping track of a given host's address proved inconvenient and prone to mistakes. In order to resolve this problem, users began giving their machines nicknames, which were easier to remember.

Soon after, a BSD Unix decided to build an automatic system (called a resolver) to map the nicknames to their ARPAnet addresses. The system was based around a single file called `/etc/hosts`. Its format was a space-separated two-column flat list, with a typical entry in the table such as:

```
18.18.0.249 vice-grips.mit.edu,vice,vice-grips
```

The first column listed the address and the second indicated the nickname(s), separated by commas. This file was downloaded from SRI's Network Information Center (SRI-NIC) and parsed. The file was subsequently uploaded to the other machines via the network or by SneakerNet¹⁴. Users simply queried the resolver for the address attached to a given machine. The resolver searched through the file for a matching name and returned the associated address.

Initially the system worked excellently. However, as the number of hosts increased, its file synchronization became problematic and resolver speed suffered as the file grew. To combat these issues, a new system called the Domain Name System (DNS) was developed to replace the host tables.

¹¹ARPA stands for Advanced Research Project Agency

¹²ARPAnet only had 8 bits to allocate for an address, and therefore only 255 addresses were possible.

¹³Internet Protocol Version 4

¹⁴*a.k.a.* putting the data on a floppy and walking it to another machine

2.3.2 Domain Name System (DNS)

The primary goal of the Domain Name System (DNS) [AL97, MD88] was to scale up the host tables, but it also included services applicable to directory look-up and email relaying. DNS is a large database that is distributed over a large number of servers. A client queries a DNS server to gather the requested information. Since a single query-response exchange with a single server would be insufficient to keep the system scalable and robust, DNS uses more than a simple monolithic client-server architecture¹⁵. Instead, DNS uses a hierarchy encoded into the DNS name to encode a sequence of servers that are queried to return an IP address.

The designers of DNS realized that their system needed to follow the basic tenets of the ARPAnet: to keep working in the face of a devastating attack. The concerns of the ARPAnet were weapons of mass destruction, such as atomic bombs. Since DNS needs the Internet to be useful, the same attacks were of concern as well as additional scenarios inspired by the previous service:

- An attacker breaking into the server and taking the entire system down.
- The number of client queries overwhelm the server.
- A rogue administrator feeding false information to the service.
- Two administrators choosing identical host-names. (Which one is valid?)
- Inattentive administrators not keeping their `/etc/hosts` up to date.

A concept that addresses many of the above problems is that of a domain. A domain is a region of responsibility and control. It also provides a mechanism for delegation of the responsibility between different parties. As an example, consider the DNS name `vice-grips.mit.edu`. DNS uses the separator “.” to indicate a domain, splitting the name into three domains: `vice-grips`, `mit`, and `edu`. These domains are ordered in a hierarchical tree structure, with the root node implied at the top. Next comes a top-level domain, then second-level, and so on. This structure is shown in Figure 2-5.

There is an implicit “.” at the right end of each name, but this is often left off for the sake of simplicity. In the actual encoding of the configuration files, a domain is only complete if the closing “.” is present. If the “.” is left off, the local domain is appended. For instance, at MIT, `vice-grips` would turn into `vice-grips.mit.edu.`, but `vice-grips.` would be left alone as shown in Table 2.6¹⁶.

¹⁵Imagine a single server responding to the estimated 300 million hosts as of January 2000 [isc].

¹⁶An amusing related effect arises when automatic appending interacts with misconfigured resolver. As an example, the typical self-referential name for a machine is `localhost`. When I plug in my machine `localhost.mit.edu` to the network, older machines become confused.

DNS Name 1	DNS Name 2	What servers are in common?
vice-grips.mit.edu.	www.objid.net.	same root server
vice-grips.mit.edu.	andrew.cmu.edu.	top level edu. server
vice-grips.mit.edu.	lovecanal.mit.edu.	mit.edu. server

Table 2.6: Table of DNS server/domain commonality

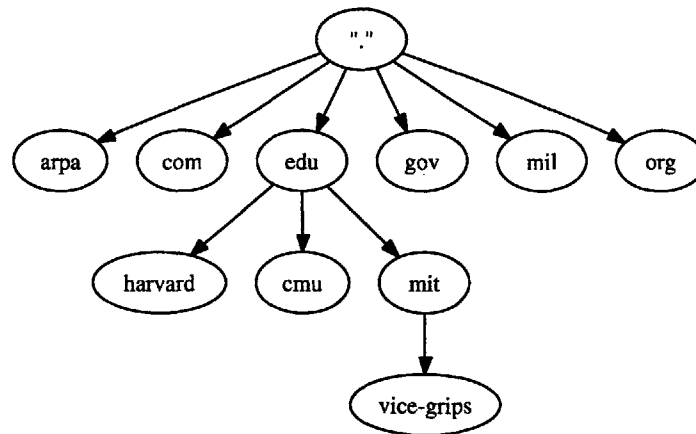


Figure 2-5: Example of a DNS resolution path [AL97]

To look up information associated with a given host name, the server starts with breaking the name into domains and beginning with the rightmost domain. Staying with our example of `vice-grips.mit.edu`, the resolving process is shown in Figure 2-5. The resolver has an internal copy of all the root servers (“.”) that is occasionally updated. After choosing a root server, the resolver then looks at the first domain `edu` and queries that server about the top-level domain’s servers. The top-level domain servers (the `edu`. servers) are then asked for information about the next domain (`mit`). Finally, the second-level servers at MIT are asked for information about `vice-grips` and they return the requested information.

To enhance performance, query responses are cached by the resolver¹⁷. Caching is made easier because DNS information is structured in a convenient manner. As the resolver recurses down the DNS tree structure, it remembers what answer it received for a particular domain. Once it has seen what the `edu` servers are, it does not need to ask again. To make sure that data is kept up to date, the concept of a Time To Live (TTL) is employed for every piece of information. When the amount of time since a local record has been accessed exceeds the TTL, the record is deemed expired and sets `DELETED`.

¹⁷Early windows resolvers did not, which was irritating as they caused a lot of traffic.

One of the beauties of the DNS system lies in the ability of a user to use a correctly configured DNS resolver with not change in their behavior, even if they were previously used to a resolver using `/etc/hosts`.

2.3.3 Hesiod

Hesiod is a service developed by MIT's Project Athena to address a need for Internet-enabled look-up system for electronic services. Servers needed to know what version of software to install and where to find it. Workstations need the locations of users' home directories, UNIX shells, and mail [Dye88]. Since DNS was always installed on workstations on workstations, anyhow, using a service such as Hesiod to meet the above needs was a logical choice.

Hesiod encapsulated the various types of information in TXT records of DNS look-ups. These records are acquired by using specially constructed DNS names arranged in a simple hierarchical format. Constructing a Hesiod request requires knowing the following pieces of information.

Hesiod Server: The `SERVER` contains the Hesiod database. It is a DNS server with carefully constructed TXT records. The Hesiod convention is to put an `ns` in front of your domain name.

Hesiod Name: The `NAME` contains the name of the entity we want to find information about.

Hesiod NameType: The `NAMEType` contains the type of information we are interested in.

Queries are constructed as:

`NAME.NAMEType.SERVER`

The result is a TXT record.

As an example, consider my workstation. It has the DNS name `sunnuva.mit.edu`. When I installed Athena on it, the installer needed to know about the machine's configuration including what software to install. This information is of the `NAMEType cluster`. From previous discussion, we would expect the Hesiod server to be `ns.mit.edu`. However, MIT added `athena` after the `ns` as it originally expected `athena.mit.edu` to be the domain.

Our DNS query would be `sunnuva.cluster.ns.athena.mit.edu`. Listing 2.1 shows what this typical Hesiod request would return. Let us analyze the return value:

`syscontrol` indicates what version of the Athena software it needs, and where the configuration file for that version lives. This workstation is version 9.4.

```
1 # host -t txt sunnuva.cluster.ns.athena.mit.edu
;; Truncated, retrying in TCP mode.
sunnuva.cluster.ns.athena.mit.edu is an alias for mrinternal\
-25458.cluster.ns.athena.mit.edu.
mrinternal-25458.cluster.ns.athena.mit.edu descriptive text "\
syscontrol control/control-9.3 9.3"
mrinternal-25458.cluster.ns.athena.mit.edu descriptive text "\
syscontrol control/control-9.4 9.4"
6 mrinternal-25458.cluster.ns.athena.mit.edu descriptive text "\
sysprefix /afs/athena.mit.edu/system/rhlinux"
mrinternal-25458.cluster.ns.athena.mit.edu descriptive text "\
lpr hole-hawg"
mrinternal-25458.cluster.ns.athena.mit.edu descriptive text "\
syscontrol control/control-8.3 8.3"
mrinternal-25458.cluster.ns.athena.mit.edu descriptive text "\
syscontrol control/control-8.4 8.4"
mrinternal-25458.cluster.ns.athena.mit.edu descriptive text "\
syscontrol control/control-9.0 9.0"
11 mrinternal-25458.cluster.ns.athena.mit.edu descriptive text "\
syscontrol control/control-9.1 9.1"
mrinternal-25458.cluster.ns.athena.mit.edu descriptive text "\
syscontrol control/control-9.2 9.2"
```

Listing 2.1: Hesiod cluster look-up

`sysprefix` is pre-pended to the highest `syscontrol` entry to determine the file-path of the install and version control files. In this case it would be `/afs/athena.mit.edu/system/rhlinux/control/control-9.4`.

`lpr` is the name of the printer that the workstation prints to by default.

There is no authentication or encryption mechanism employed by Hesiod, since standard DNS is used. The information returned is not particularly useful to most attackers. Knowing the version of the software run on a given workstation could indicate particular vulnerabilities to an attack but the Hesiod information does not include the sub-version numbers, which are the ones that tend to change when bug-fixes are implemented and exploits are closed. The example workstation is version 9.4.22.

The mechanism for logging into an Athena workstation requires user related information. In particular, the password info, home directory, email server and group memberships are needed. The Hesiod user look-up request is illustrated in Listing 2.2.

```
# host -t txt foley.passwd.ns.athena.mit.edu
foley.passwd.ns.athena.mit.edu descriptive text "foley\
*:7812:101:Joe Foley,Crazy old Joe\
,35-010,2588547,6172256696:/mit/foley:/bin/athena/tcsh"
# host -t txt foley.filsys.ns.athena.mit.edu
foley.filsys.ns.athena.mit.edu descriptive text "AFS /afs/net.\
mit.edu/user/foley w /mit/foley 2"
foley.filsys.ns.athena.mit.edu descriptive text "AFS /afs/sipb\
.mit.edu/user/foley w /mit/foley 3"
foley.filsys.ns.athena.mit.edu descriptive text "AFS /afs/\
athena.mit.edu/user/f/o/foley w /mit/foley 1"
# host -t txt foley.pobox.ns.athena.mit.edu
foley.pobox.ns.athena.mit.edu descriptive text "POP P014.MIT.\
EDU foley"
# host -t txt foley.grplist.ns.athena.mit.edu
;; Truncated, retrying in TCP mode.
foley.grplist.ns.athena.mit.edu descriptive text "tepazoo\
:49112:biotgc:37697:ko-gms:36258:jitsu-off:35977:legends\
:30433:arctan-dat:30026:autoid:28737:aidc-www:28289:31337-\
d00dz:27763:sik-bastards:30575:xcom:26822:net-work:26468:\
acme-old:33867:cats:27183:apo-actives:26037:nexwar:25924:\
apo-family-jefreypu:" "25709:miths:25659:rhetoric-gms\
:26373:fsck-politics:25094:teach-physics:25092:black-ops\
:24444:glue-sniffers:24199:easyrider:23809:tech-squares-\
club-ac1:22514:riftek:22414:"
```

Listing 2.2: Hesiod user look-up

2.3.4 Object Name System (ONS)

The Object Name System (ONS) is DNS applied to IDs in general. Specifically, ONS is focused on EPC-encoded RFID tags and resolving this ID to an application. ONS accomplishes this by using DNS and encoding the ID's in a manner inspired by Hesiod [Fol99]. When an ONS query is completed, the result is information related to the ID and the IP address of servers that contain additional information.

2.3.4.1 ONS Resolution

To use ONS with a 96-bit RFID tag, we must first convert the binary representation into a format that is more acceptable for the DNS resolution. Conveniently, DNS uses domains to encode when responsibility is handed off between servers. EPC RFID tag also encodes a similar division but the responsibility was defined by the standard at that time.

In 2001, ONS was turned over to Verisign to develop by EPCglobal. One of the major changes made to the system under the new partnership was the creation of additional information available for extraction out of an ONS query. My original version had all EPC-IS queries and tag divisions the same for simpler implementation, but such a system would have quickly become obsolete. In ONS 0.9, developed with assistance from Sridhar Ramichandran of OATsystems, a pre-resolver was added to dynamically encode and format the EPC to be used in ONS. This pre-resolver decided where tags would be divided on a per-tag basis via NAPTR records¹⁸. These records contained formatting strings, very similar to the *pack* and *unpack* functions in perl [WCO00]. In addition, the NAPTR record would give hints as to what types of queries were valid at the EPC-IS server and how they should be formatted. A concern was raised that the additional data might make syncing everything harder and increase the load on ONS and its database, but this concern was alleviated thanks to DNS caching.

This system was deemed too complicated and the latest ONS standard (1.0) passes the responsibility of bit encoding to a separate standard called the EPCglobal Tag Data Standard [EPC06]¹⁹. The standard specifies a fixed set of encodings and divisions according to the first 8 bits. Recall the example of some of the encodings in Table 2.2. The GID-96 standard detailed in the Tag Data Standard divides a generic 96-bit tag along these divisions (Table 2.7).

To illustrate how the current ONS pre-resolvers operate, consider an EPC GID-96 tag encoded the information shown on the bottom of Table 2.7 (decimal representation).

¹⁸a type of record in DNS

¹⁹which has striking similarities to my original proposal

	HEADER	GENERAL MANAGER	OBJECT CLASS	SERIAL NUMBER
	0011 0101	28 bits	24 bits	36 bits
<i>Example</i>	<i>53</i>	<i>101</i>	<i>7812</i>	<i>277</i>

Table 2.7: GID-96 bit divisions

HEADER	0011	0101							
GEN. MANAGER	0000	0000	0000	0000	0000	0110	0101		
OBJECT CLASS	0000	0000	0001	1110	1000	0100			
SERIAL NUMBER	0000	0000	0000	0000	0000	0000	0001	0001	0101

Table 2.8: Read result for an example EPC GID-96 tag

An ONS pre-resolver operates thusly:

1. The tag ID is read.
2. The tag ID's bits are divided (Table 2.8) according to information given by the header (Table 2.2).
3. The divided bit sequences are hex encoded (Table 2.9).
4. The hex encoded divisions are placed in an URI²⁰ separated by periods. The URI header indicates the disposition from the ID's header:
urn:epc:id:gid:0000065.001e84.000000115.
5. The URI is converted into a DNS name:
000000115.001e84.0000065.gid.id.onsepc.com²¹.
6. The resolver makes a DNS request for any NAPTR records associated with that DNS name.
7. The NAPTR record contains URLs to servers for associated information.
8. The application accesses the URLs using separate web services.

The look-up operates in the same manner as DNS (also shown in Figure 2-6):

1. The machine goes to the root name servers ("."), which return the com. servers.
2. A com. server returns the onsepc.com. servers.

²⁰Uniform Resource Indicator

²¹Previously, ONS operated in the objid.net domain.

HEADER	3	5							
GEN. MANAGER	0	0	0	0	0	6	5		
OBJECT CLASS	0	0	1	e	8	4			
SERIAL NUMBER	0	0	0	0	0	0	1	1	5

Table 2.9: Hex encoding result for an example EPC GID-96 tag

3. A `onsepc.com`. server returns the `id.onsepc.com`. servers.
4. A `id.onsepc.com`. server returns the `gid.id.onsepc.com`. servers.
5. A `gid.id.onsepc.com`. server returns the `0000065.gid.id.onsepc.com`. servers.
6. A `0000065...` server returns the `001e84.0000065...` servers.
7. A `001e84...` server returns the `000000115.001e84...` server.
8. We then have IP address of the server that has information on that tag.

The IP number is stored using either a standard DNS server. The machine would then query that server in some standardized format such as HTTP or SOAP.

2.3.4.2 Star-Records

An apparent problem with configuring an ONS server for many different IDs is managing the large number of repeating destinations for a set of IDs. Storing a record for every ID is an inefficient waste of space. Conveniently, DNS has star-records, meaning number aggregation was already inherent in the file format. As an example, a record that was specified as:

`*.EE.FG.objid.net`

would match:

`00.EE.FG.objid.net` and `01.EE.FG.objid.net`

allowing both records to have the same address in a compact notation. ONS makes use of these star-records to group together IDs that share an EPC-IS server.

2.3.4.3 Demesnes

During his talks on The Onion Router (Tor) [Tor06], Roger Dingledine often repeats the phrase: “The Internet is not flat.” What Roger means by this is that context sometimes changes the content. As an example, if you access `google.com` from an address that has a DNS name associated with it that ends in `.de`, you will get a page

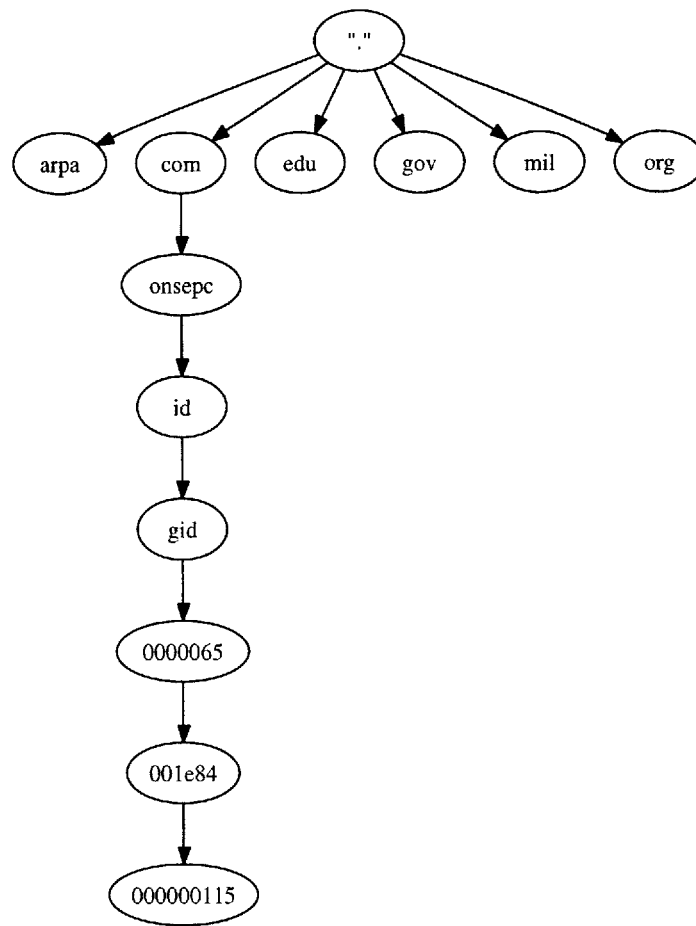


Figure 2-6: ONS resolution example

in German, regardless if your browser has given any information about what language you prefer.

OATsystems helped develop this concept in ONS. The concept of an internal and external view which uses this concept of names indicating location was designed and implemented.

Many corporations wish for information to be viewed differently from within the corporation and the outside world. This is often accomplished using a Virtual Private Network (VPN) to prevent access to parts of an internal network except via a special tunneling client. The concept is applied to the remapping system described as part of TagCheck in Section 5.1.2.3.

Together with OATsystems we worked on making ONS usable in the context of an external/internal view system. OATsystems created a configuration system that generated the external and internal views that we desired, with certain parts of an ONS tree visible only if the requester were in an authorized IP space. A standardized INVALID DNS NAME error was returned to unauthorized clients.

The internal request names would be composed in the following way:

`SERIAL.MODEL. internal.MANUFACTURER.VERSION .epc.objid.net.`

Compare this to a standard request:

`SERIAL.MODEL.MANUFACTURER.VERSION .epc.objid.net.`

The addition of the internal marker makes it very easy to modify application code to make use of the differentiation.

2.3.5 UCCNet

A system with a similar purpose as ONS but different focus is UCCNet [1Syb, 1Sya]. It is designed as a glue protocol and database between suppliers and retailers. For instance, if Sears wants to produce a new catalog, they need information about the products: costs, target consumer, and product pictures or artwork. This becomes even more necessary in the world of Internet shopping, where prices are constantly changing.

For a subscriber fee, UCCNet will setup a manufacturer's site with their software. The software implements a fairly general XML DTD which encapsulates the content that the manufacturer or supplier wants to provide. The information is converted into another internal representation and stored in a database. The retailer pays a fee to query the database for the information they desire.

This system does a good job of describing objects in the retail world. However, lacks the the concept of instances we require for the more interesting applications mentioned later. It is also very centralized, which means it may not scale well for our purposes. The major disadvantage for general application of such a system is that access is not free to the database; this is acceptable for retails, but not the average consumer.

2.3.6 Handle

Handle [Hana] is a different type of object naming system created by CNRI²² in 1994, to map IDs to a persistent network identifier. [DOI], who wanted a unified system capable of associating unique ID's to articles that could be tracked indefinitely [Hanb].

The Handle system satisfies many of our requirements as well. It creates a unified network database that will return information when given a unique ID. The IDs are of the form:

NAMINGAUTHORITY/IDENTIFIER

However, unlike DNS, Handle only has a two-level hierarchy, as should be apparent from the ID style. It uses a combination of a hashing algorithm and a cluster of servers to create a load-balanced, robust, and scalable service. Handle makes use of a concept called a Dynamic Hash Table (DHT) which uses a hash function to deterministically assign and locate IDs evenly to a cloud of servers.

There are interesting questions about the management of such a system and the limits of its scalability. One major challenge is that Handle software is not considered part of the standard software package installed on a computer. This service seems to have made significant penetration in the area of literature identification, but not other fields.

2.3.7 Hybrid Resolution

In [Doi05], Doi noticed that there are particular design aspects that make DNS and Dynamic Hash Table (DHT) popular as discovery mechanisms for RFID. The hierarchical basis of DNS makes it very scalable within the division of its domains. It also encodes the tag in a manner easy to understand by humans and computers alike. DNS is heterogeneous and therefore applicable to a variety of systems since the name-space may be reallocated easily. Finally, DNS is legacy-friendly, as most modern computer systems and applications have the ability to use DNS without a separate dedicated subsystem.

DHTs have a better ability to scale when used with data that has a flat name-space, such as RFID tag serial numbers. DHTs are more flexible in terms of scalability – adding capacity does not require careful consideration of all the structures involved. DHTs are also easier to maintain than DNS, as the algorithmic nature of the way IDs are handled is autonomous, requiring little user or administrator intervention.

Doi suggests that the proper way to leverage the strengths of DNS and DHT systems is to build a hybrid system. He proposes using DNS for discovery of the header, general manager and object class since they encode data in a structured manner. He then suggests using DHT for the serial number field as it is a flat name-space. To allow these two systems to seamlessly interact, Doi's design uses a

²²Corporation for National Research Initiatives

translation service that emulates a DNS server but is effectively a part of a DHT network.

Doi's design is a good solution to some aspects of the data management problem. However, it only seems necessary if the serial numbers for a given object class are located on many different servers. For most products at a basic level of service, this situation is unlikely. Most items of a given class will probably have their information stored on the same EPC-IS server. In this case DNS with star-records²³ is very efficient and very fast. Doi's solution will be needed during transference of ownership, which causes the manufacturer to no longer be the authoritative data source. In this case, the serial number becomes very important for finding relevant data. My proposed system, TagFolio, addresses change of ownership in a different manner (Section 5.1).

2.3.8 DNSSEC

In 1990, Bellovin [Bel95] discovered a major security hole in DNS that would allow an attacker to easily break into a host by hijacking parts of DNS [Gie]. His discovery was that the weak reverse-resolution binding mechanism of DNS that allowed access to machines using the trusted host mechanism of `rlogin` and `rsh` could be attacked.

To eliminate this exploit, Bellovin developed DNSSEC, specified by IETF²⁴ RFC²⁵ 2065, 2535, and currently 2535bis. DNSSEC [dnsb] uses cryptographic signatures to ensure that information is coming from an authoritative server. This system is known as an authentication chain.

The Root Name Servers each have a well-known public key pair (DNSKEY RR) used to sign the keys of the top-level domain servers with a Delegation Signer Resource Record (DS RRSet). DS RRSets are authenticated by DNSKEY RRSet, and contain the hash of another DNSKEY RRSet for the sub-domain controlled by the given domain server. This protocol creates a straightforward procedure for validating the domain. A client looks at the signatures accompanying a query to determine if the correct path has been followed. Conveniently, the signatures are cached, so they do not need to be looked up every time [DNSa].

DNSSEC is excellent for ensuring that information is valid and that the chain of authority has been properly followed, but it doesn't address the issue of privacy. No encryption is specified, only authentication. In order to build a secure system capable of ensuring privacy, we need something more along the lines of TorONS, which will be discussed in Section 7.3.2.

²³described in Section 2.3.4.2

²⁴Internet Engineering Task Force

²⁵Request For Comment - preliminary Internet protocol specifications for peer review

2.4 Privacy Policy Negotiation

A major challenge in privacy and security is defining where the boundaries lie between the various shades of privacy. In order to determine if a particular action complies with a policy, the agents must all be able to understand how the various elements interact. In electronic media, such policies must be encoded in formats able to be parsed by a computer.

2.4.1 Platform for Privacy Practices (P3P)

Platform for Privacy Practices (P3P) [p3p] was designed specifically to describe a data-collection agent's intent for web-based applications, focusing primarily personal data requested by websites. P3P attempts to encode the intent of the data gathering party in a format that humans can easily understand and applications can enforce.

The aspects covered by P3P(From [p3p]):

- Who is collecting this data?
- Exactly what information is being collected?
- For what purposes?
- Which information is being shared with other parties?
- Who are the other data recipients?
- Can users make changes to how their data is used?
- How are disputes resolved?
- What is the policy for retaining data?
- Where can the detailed policies be found in "human readable" form?

To use P3P, a site administrator generates a P3P file to place on the website. When a user accesses the website, the P3P-enabled application fetches the P3P file from the website and examines its contents. If anything in the P3P file disagrees with the privacy practices the user has enabled in their browser, the user is warned. This allows the user to pass public information to a server with minimal effort, while restricting private information transfer unless specifically authorized.

2.4.2 UbiComp Privacy

Wu *et al.* discusses the need for privacy policy distribution and verification in an Ubiquitous Environment (UbiComp) [MW02]. The system they proposed in the paper applies the Platform for Privacy Practices (P3P) to a UbiComp platform. As an example, a user would travel with their P3P Enabled PDA, which would inform any wireless systems nearby with this policy. The policy would then be parsed and applied to the interaction with the owner. The proposed system shares interests with TagFolio, discussed in Section 5.1.

This is an excellent idea, but it requires that all applications be able to parse the P3P format and respect its wishes. There is no way to enforce the policy because verification is challenging. A level of trust is needed beyond a traditional interaction.

2.4.3 Data Guardian

Rieback *et al.* describes a general-purpose RFID privacy assistant platform to be integrated into PDA's and cell phones called the RFID Guardian [RCT05]. The RFID Guardian combines 4 security functions into a single unit: auditing, key management, access control, and authentication. The auditing feature detects reads by other readers and thus can notice changes in the number and identity of tags associated with a user. Key management involves keeping track of the keys and kill codes associated with the user's tags.

The Guardian manages access control in three ways. It is able to coordinate the security primitives the user has assigned to their tags, making sure that every tag knows what it should and should not be doing. Second, with its increased ability to gather information and save state, the Guardian can be context aware enough to tell tags when the environment has changed and their operation should switch to a new mode. Finally, the Guardian can propagate policy updates to new tags, or even change the policies on existing ones.

Authentication works by having the Guardian authenticate to the readers on behalf of the RFID tags. This also allows for proper key exchange protocols and strong cryptography to be used.

This concept of the Guardian is similar to the TagFolio design I will discuss in Section 5.1. However, TagFolio focuses on the protocol and interface necessary to provide an integrated privacy solution that continues to work after the device has left the presence of the RFID tag.

2.4.4 Personal Information Carrier (PIC)

In 1999 the military began testing electronic dog-tags. These dog-tags are an environmentally hardened secure digital flash memory card with 128MB of memory. Each soldier is issued one such tag. When a medical officer needs to access the soldier's medical records, he does so with a Personal Information Carrier (PIC) [Mur99] able

to decrypt the information on the tag. VeriChip proposes to replace dog-tags with their implanted RFID chips on military personnel [Jon06]. Such a system would need a PIC with an VeriChip reader. Further discussion of RFID-enabled dog tags is in Section 3.3.1.1.

2.4.5 RFID Acceptor Tag (RAT)

Soppera *et al.* proposes a similarly motivated but differently composed system: the RFID Acceptor Tag (RAT) [SB06]. This system attempts to anonymize tags and provide a transaction-limited access control via specialized PRNG²⁶-generated pseudonyms produced by interactions between the RAT and the tags it controls. Using multiple writable secrets on a tag, the RAT is able to produce a “delegation tree” that allows it to efficiently map pseudonyms to IDs. This system removes the need for a reader to have a secret to gain access to a tag. Instead, the RAT acts as a broker of information. The temporary secrets can also be employed to transfer ownership with a PIN and nonce. The disadvantage to the RAT is that it stops protecting tags when it is not present. There is also a significant overhead associated with maintaining the delegation tree.

2.4.6 Watchdog Tag

Floerkemeier *et al.* describe a privacy verification method using an innovation called a Watchdog Tag [FSL04]. This active device takes an individual’s privacy preferences and shows if reader transactions in the area comply with these preferences or not. Such a device is the logical RFID progression of the UbiComp system described in [MW02].

2.5 Access Control

The domain of access control is most important at the application level. Once the resolution data is acquired, transmission of application data is possible. The next challenge becomes to determine who should have access to the various elements of the data. Universal open access is clearly not the answer, as too much information is exposed to possibly malicious users. Even assuming an access control solution exists, administration of the multiple different entities accessing it and the changing nature of their permissions is challenging.

²⁶PRNG

2.5.1 SIREN

The SIREN system developed by Singh is a design producing a generalized framework within SQL (Simple Query Language) to allow a database to have Roles-based Access Control [Sin02].

Many computer access control systems think of access in terms of servers and connections. SIREN realizes that the base currency that needs to be controlled is the data and therefore the database itself.

SIREN uses a separate internalized database examined during initial access. This database is parsed to determine a given user's set of roles. Once these roles are found, they are translated into filters. When the user attempts to access a given data set, their request is processed through the set of filters. Requests for changing data are processed through a related set of filters generated from the same rule-set. This ensures that the user has access to only the data allowed for both reading and writing.

The major benefit to a system such as SIREN is the transparency of its access control. The user's queries are simply transformed and processed using a standard SQL back-end. The only element that needs to be added is the middle-ware SIREN layer between the application and the SQL server.

2.5.2 Enterprise Privacy Authorization Language (EPAL)

Enterprise Privacy Authorization Language (EPAL) is another project sponsored by the W3C to formalize privacy preferences [AHK⁺03]:

“An EPAL policy defines lists of hierarchies of data-categories, user-categories, and purposes, and sets of (privacy) actions, obligations and conditions. User-categories are the entities (user/groups) that use collected data. [...] Data-categories define different categories of collected data that are handled differently from a privacy perspective. [...] Purposes model the intended service for which data is used [...]” [AHK⁺03]

EPAL is designed to be used similarly to P3P, but is a more general concept. P3P defines preferences for a single entity, focused particularly on a web browser. EPAL derives entities from categorizations and the hierarchy defined in a policy and applies them to construct a data access control policy.

2.6 Federation

A concept applicable to managing changing situations and large numbers of users is federation. This concept is already used in the physical realm when one government wants to send a representative to another. They send the diplomat with difficult-to-forge papers certifying his identity, and the authority given to him.

Federation encapsulates the same relationship but in the electronic realm. Within the local control of a federation, it performs as any other traditional access control. An entity is given a certain number of privileges. The federation's special abilities are activated at the edge of the local control. If a user wishes to interact with someone in another federation, they must have their federation negotiate the details with the destination federation. Once these negotiations are complete, the user is able to use their access privileges, modified by the negotiations, in the new federation. Much of the challenge lies in making the negotiation automatic and yet rich enough to describe the various different ways that federation entities might interact.

2.6.1 Liberty

A well-publicized federation solution is the Liberty Alliance Project [CJHT]. This protocol is focused primarily on web applications. It requires a web browser (or equivalent) in order to function; many of the login systems assume that a web-redirect capability is available.

The primary goal of the Liberty Alliance is a federation system whereby a given entity can be given permissions in a variety of different web environments. The primary and initial challenge to building such a system is creating a network of trust so that the user may access a Single Sign-On capability (SSO). In such a system, a user authenticates to an initial login server, which keeps track of their credentials. The login server then forwards the credentials to other systems, which trust its authority.

One obstacle to Liberty's usability is the need to verify the federation each time an entity wishes to make use of the SSO. The security reasons are clear — you need to ensure the user knows that their credentials have been activated at a remote site. The frequent verifications add headache and complexity that will confuse the average user.

2.6.2 Kerberos

Kerberos is a trusted third-party authentication system developed at MIT as part of Project Athena. It is a comprehensive authentication and encryption package that is included in the modern versions of the Windows operating system. Kerberos understands the concept of federation and has been designed into the standard [Ker]. It makes use of a third-party intermediate called the Key Distribution Center (KDC). The KDC contains passwords of everyone associated with a given Kerberos realm. Realms are federations and map easily into the name-space. The entities that interact with a Kerberos realm are called principals.

My Athena Kerberos Version 5 principal is:

`foley@ATHENA.MIT.EDU`

This is actually shorthand for:

`foley/null@ATHENA.MIT.EDU`

Kerberos principals are broken into a number of different pieces:

`foley` refers to my user name, which I use to log in to a workstation.

`null` refers to an instance. Instances allow users to have a number of different environments to work in, which may or may not come with varying privileges. Instances use separate passwords from each other. By default, the instance is “null” or blank. A common instance used by people who access important services is to have a “root” instance. Thanks to instances, power-users can choose which environments to access at a given time. For example, they would log in to public workstations using the null instance in order to protect the critical services from malicious programs or user mistakes.

`ATHENA.MIT.EDU` is the name of the realm. This is typically a name related to the group involved. Since project Athena is what started Kerberos, the default that MIT people use is `ATHENA.MIT.EDU`.

To use Kerberos, you generally execute `kinit` to authenticate (or other dedicated login program such as `xlogin`), which is comprised of the following steps:

1. You type in your user name.
2. The login program contacts the KDC with your user name.
3. The KDC returns a special “ticket-granting ”ticket encrypted with your password.
4. When you type in your password, the program tries to decrypt the ticket using your password as the secret.
5. If the decryption succeeds, you then possess a ticket, which can be used to authenticate to a number of services including email, file-systems, and remote logins.

A particularly useful feature of Kerberos is Single Sign-On mentioned in Section 2.6.1. A given Kerberos user only needs to type in their password once every time they log in, unless their tickets expire. This is even true when logging into a remote host, if the remote access program is configured to “forward credentials.”

Interestingly enough, Kerberos also has a concept of federation. Federation is called cross-realm authentication under Kerberos; the various areas of responsibility are called realms, as mentioned above. In order to make use of the cross-realm authentication, a user simply appends the realms to the end of the principal. For instance, if I desired to authenticate to RPI with my MIT principal, I would log in as:

`foley/null@ATHENA.MIT.EDU@RPI.EDU`

Assuming that the appropriate Kerberos key sharing had taken place and that the RPI realm had some concept of what an MIT person should have access to, the system would work seamlessly.

For a more practical example, let us suppose that I am a salesperson for Procter & Gamble (P&G), I want to access my inventory information at Walmart. Under the Kerberos model I have just described, I would login to the Walmart server as the following principal:

`foley/sales@PG.COM@WALMART.COM`

The authentication mechanism will proceed to check my PG.COM credentials and notice that my instance is sales. It will then grant me access to sales information related to PG.COM articles.

Unfortunately for RFID applications, straight Kerberos is too heavyweight of a protocol, requiring a reliable network connection, strong nonce generation, and the ability to maintain an accurately synchronized clock.

2.7 Trust Model

Many of the designs mentioned in this thesis involve building a system of trust between the readers, tags and servers. A survey of trust models can be found in [PBM05]²⁷. Creating trust is often a very challenging affair as mentioned by Beal *et al.* [BS05]. They describe one of the major weaknesses in the certificate-based trust mechanism, taken from the Microsoft Internet Explorer documentation:

“... a single bad Certificate Authority completely negates the effectiveness of the entire system of Internet security.”

The centralized nature of the Certificate Authority public key infrastructure means that a single point of failure is hard to recover from. Declaring that a given public key pair will never be compromised is folly.

2.7.1 Web of Trust

The model used in encryption programs such as Zimmerman’s Pretty Good Privacy (PGP) is that of a “web” of trust [Zim94, Gar94]. A given member is defined as a set of public keys associated with a name such as an email address. My PGP identity is shown in Listing 2.3.

²⁷a chapter in [CG05]

```

# pgp6 -kvv foley
Pretty Good Privacy(tm) Version 6.5.8
(c) 1999 Network Associates Inc.

Export of this software may be restricted by the U.S. \
government.

Looking for user ID "foley".
Type bits      keyID      Date      User ID
DSS  1024      0xE8665A5F  2002/11/18
DH   3072      0xE8665A5F  2002/11/18 *** DEFAULT SIGNING KEY \
***
                                Joe Foley <foley@mit.edu>
sig                                0xE8665A5F      Joe Foley <foley@mit.edu>
RSA  1024      0x4F7ABB79  2002/11/18 Joe Foley <foley@mit.edu>
sig                                0x4F7ABB79      Joe Foley <foley@mit.edu>
2 matching keys found.

```

Listing 2.3: Example of a PGP key listing

Each key pair has a keyID is calculated using the hashed 32-bit value of the public key. The number of keyIDs between collisions by a uniformly distributed b -bit²⁸ hash generations can be calculated via the birthday paradox in [Sch96]:

$$n = 2^{\frac{b}{2}} = 2^{\frac{32}{2}} = 65536 \quad (2.3)$$

This is insufficient for large scale applications, but for the small user-base that PGP is targeted towards, it suffices.

Web of trust defines the interactions between keyIDs in a manner similar to human social interaction. A keyID may sign another key-pair with its private key, with the intent of declaring: "This key pair is associated with the person that is mentioned in the description." This signature, in an electronic form, is then published on a PGP key-server²⁹ or a public website where a typical user can find it. When someone downloads the signature and installs it using PGP, the signature attaches to the public key pairing in their key-ring³⁰. Depending upon its version, PGP uses this information in various ways. In the earlier versions of PGP, such as 2.6.2, the use of signatures was purely informative for the user. To see if a particular keyID was a correct match for a person, the user examined the list of signatures to see if they personally trusted any of the signers with respect to the identity of the owner.

In the newer versions(≥ 5), PGP automates this process by adding a user-selected

²⁸where b is 32

²⁹a specialized web server that maintains a database of public keys and signatures

³⁰where a user's keys are stored

“trust” value to any key when it is added to their key-ring. This value can be set from “Don’t Trust” to “Trust explicitly on all matters.” When subsequent keys are added to the key-ring, PGP first checks to see if the keys were signed by any of the already trusted keys, if so the new key is automatically assumed to be valid without further prompting. Thus the web of trust is built by having people act as introducers and mediators. This process is analogous to indirect human trust mechanism in the real world. Beal *et al.* proposes a similar but more light-weight mechanism for creating a trust model discussed in Appendix A.6 [BS05].

2.8 Key Predistribution

A major challenge with any cryptographic system is setting up the keys. In a symmetric system, the keys must be securely distributed to both ends. In a public cryptography system, a method of ensuring that the public key at each endpoint is correct is needed. This problem is called key predistribution. In this section, I present various solutions proposed for addressing key predistribution.

2.8.1 Resurrecting Ducklings

Stajano proposes a very simple model for key setup in [Sta00, Sta02]. In his model devices are represented as ducklings. When a device is newly created as a newborn duckling, it needs to be imprinted. When it meets up with a device desiring control, it imprints to that “mother”. The mother downloads configuration information including an authentication key to the duckling. Stajano calls this information the “soul”. Subsequently, the duckling will only talk to and obey commands from its mother. This continues until the duckling “dies” and resurrects as a new duckling, ready to be imprinted. The mother is the one to give the suicide command.

2.8.2 Du-Deng-Han-Varshney (DDHV)

An interesting approach to solving the public key infrastructure problem is proposed in [DDHV06] using a system called the Du-Deng-Han-Varshney (DDHV) scheme. The proposed system is designed for a sensor array network, but it is equally applicable to reader arrays.

DDHV uses a symmetric algorithm mentioned in [Blo84] that allows any pair of nodes in a network to derive a pairwise key. In addition, as long as no more than a selected number of nodes (λ) are compromised, the links remain secure. DDHV uses the work of Blom *et al.* to make use of the deployment topology to increase the chances that neighboring nodes share keys [Blo84]. Details of the matrix math creating this property are beyond the scope of my thesis.

2.8.3 Eschenaur-Gligor (EG)

[EG02] divides key agreement/distribution systems into three types:

Trusted server: A third-party server is used to manage the keys. This is problematic with sensor array systems because there is no trust infrastructure in the cloud of sensors.

Self-enforcing: Asymmetric cryptography is used to set up connections between the sensors. This unfortunately requires computation and energy resources that may not be available or cost-effective for RFID applications.

Key predistribution: The keys are placed in the sensors before they are deployed.

A naive key predistribution system would place a master secret on all sensors. This is problematic because a single compromise anywhere in the system will compromise all devices. Making the secret hard to retrieve from the devices only staves off the inevitable in an insecure environment³¹.

A better key predistribution is the Eschenaur-Gligor (EG) scheme [EG02], which breaks the task into three segments:

Key predistribution: Each node selects τ keys from a very large pool of keys (S), forming the sensor's key-ring. $|S|$ is chosen so that the probability that one key is shared between two nodes is p .

Key setup: Each node looks for neighboring nodes with the same secrets. As they do so, a graph of the secure links is built.

Path key establishment: Once the secure links are established, the nodes that do not share keys have separate path keys set up by routing a request for a key exchange through the secure links.

The challenge with this scheme is to choose the right $|S|$ for a given network. Too few keys mean less effort is required to compromise significant parts of the network. Too many keys lead to dropped connectivity of the nodes. Thus, $|S|$ will vary according to the application and results of field-trials.

2.9 Anonymization

Mobile devices able to access a large network transparently tend to leak or actively provide location information. Such information may be used to track the possessor, and possibly further violate their privacy. Gorlach *et al.* describe four possible attacks on mobile devices [GHT04]:

³¹though PUF [GCvD⁺02b] systems may make this less of a problem

First-hand communication: The device is tricked into allowing private information to be released by direct communication.

Second-hand communication: Private information is relayed to another party by way of an unauthorized individual.

Observation: The attacker gains information by simply listening without taking an active role.

Inference: This is a multi-media attack that uses the leaked information, correlating it with other data to come up with hypotheses about what is going on.

2.9.1 Cricket

Gorlach *et al.* catalogs a number of possible solutions to the above threats ranging from policies and government intervention to more technical solutions [GHT04]. The most interesting of these solutions is called the Cricket Location-Support system. This system instructs the device not to transmit information in order to gain location information. Rather, the device listens for various clues to resolve its location, transferring the responsibility of location state updates to the device.

2.10 Summary

In this chapter I have presented various RFID security-related technologies. These different elements complete a survey of the background research leading to the developments in this thesis. The concepts discussed will be highly relevant to the development of the TinFoil RFID protection architecture, as will be discussed in the next chapter.

Chapter 3

System Design of a Secure Identification System

“The first rule of fight club is that you do not talk about fight club.”[Pal96]

To identify the requirements of a security model, we must analyze the system as a whole. This requires us to understand both the model itself and its application(s). The security needs of a maximum-security prison are different than those of a preschool playground. It does not make sense to use the same security model for both cases, even though the end goal is essentially the same: keep the occupants in, and the outsiders out.

This chapter creates a framework for my work to fit in and describes the elements needed to complete it. I begin by examining the target audience and specifications of electronic security. Next, I examine methods for analyzing security. I proceed to specify and design some of the basic concepts necessary to create a secure system. I finish by constructing a framework, TinFoil, for all the elements of this research to integrate into.

3.1 Defining Security Itself

The first step in evaluating any design is to understand the issues being addressed. For my part, I wish to focus on privacy and security applied to RFID in this thesis. Unfortunately, it is not easy to quantify privacy and security. At the US National Science Foundation’s 2005 Cyber Trust meeting recanted in [Lan05b] a concept of green and red environments was presented. A red environment is a risky one, freely accessible by the Internet which brings with it much curiosity and maliciousness. A green environment allows greater protection by restricting communication with the outside world.

Bishop *et al.* presents a similar concept of security being envisioned as a castle keep [BF04]. The goal of security administrators is to straighten the castle wall and ensure

that only authorized individuals are allowed to enter and access the castle's resources. A major flaw of this vision is that the most damaging attacks often come from the inside. A highly secure computer facility can be penetrated by an authorized user bringing a Trojan horse program into the system, so named for the mythical wooden horse full of Spartans at the fall of the Greek city of Troy.

We can base our confidentiality concepts on certain government regulations. The familiar definition of secrecy levels based on damage of disclosure is described in Executive Order 12958 [Hou95]:

Sec. 1.3. Classification Levels. (a) Information may be classified at one of the following three levels:

1. "Top Secret" shall be applied to information, the unauthorized disclosure of which reasonably could be expected to cause exceptionally grave damage to the national security that the original classification authority is able to identify or describe.
2. "Secret" shall be applied to information, the unauthorized disclosure of which reasonably could be expected to cause serious damage to the national security that the original classification authority is able to identify or describe.
3. "Confidential" shall be applied to information, the unauthorized disclosure of which reasonably could be expected to cause damage to the national security that the original classification authority is able to identify or describe.

In addition, the term "Unclassified" is applied to any document less secret than "Confidential". Such documents are considered harmless.

Another set of definitions, this time from the military realm, is of the level of visibility of an operation [Sta06]:

Clandestine: The operation that takes place in total secrecy. The enemy is not to know that it even took place.

Covert: The operation takes place with some amount of discretion. The enemy may gather some information about what is going on, but not enough to compromise the operation's success.

Overt: The operation may be largely visible to the enemy.

Using both sets of classifications as a basis, I propose the following set for use with industrial applications:

Arcana: information that could bring about the total ruin of the owner or his associates if disclosed inappropriately

Private: information that could cause large monetary, mental, and/or physical harm to the owner or his associates if disclosed inappropriately

Intimate: information that could cause significant discomfort and small financial damage to the owner or his associates if disclosed inappropriately

Common knowledge: all other information

Another valuable, though seemingly obvious concept from [Hou95] is that of a security level water mark. A discrete unit of data is always deemed of the security level of the most restrictive classification associated with any single element within the unit. For example, an “Intimate” document becomes “Private” if even a single “Private” piece of information is added.

The concept of levels is considered somewhat antiquated in the current Top Secret specifications. The modern trend for sensitive information is to compartmentalize access. This style of controlled access is denoted with the code “SAP” for Special Access Programs. Projects that have an SAP clearance are not available to Top Secret clearance individuals unless given specific project access [Hou95]. This level of access was mostly subsumed by the classification of Sensitive Compartmented Information (SCI) [oD01].

In addition to security, we need to codify attitudes towards privacy. These are best grouped into categories described by Columbia University Professor of Public Law Alan Westin from a series of surveys he did in 1999 [dJ06]:

“These are Privacy Fundamentalists, who reject offers of benefits, want only opt-in, and seek legislative privacy rules; Privacy Unconcerned, who are comfortable giving their information for almost any consumer value; and ... Privacy Pragmatists. Privacy Pragmatists ask what’s the benefit to them, what privacy risks arise, what protections are offered, and do they trust the company of industry to apply those safeguards and to respect their individual choice.”

In his investigation, Westin discovered that consumers had “become quite privacy assertive” in their interactions with businesses. In many cases, the decision of whom to do business with was dictated by the company’s privacy policies. In later testimony, Westin stated that approximately 125 million people could be classified as Privacy Pragmatists [Wes01]. This large number indicates that privacy in commercial transactions has become a significant factor.

3.2 Threat Analysis for Identification Systems

In a perfect world for security professionals, a good off-the-shelf solution would exist for any security or privacy problem. Such a solution would unfailingly allow

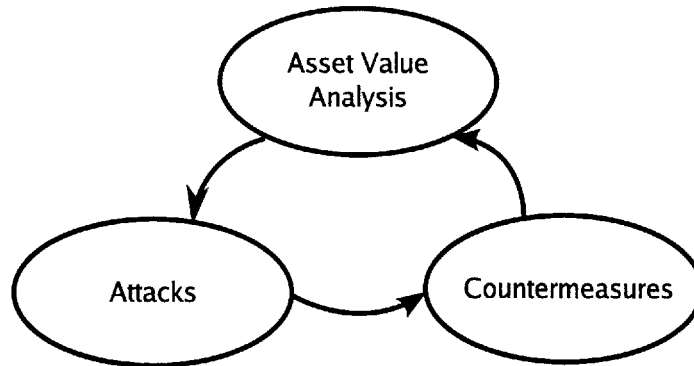


Figure 3-1: The struggle of security

only authorized entities to access data. Unfortunately, we do not live in a perfect world. Instead, we have many solutions, each with its own individual advantages and disadvantages. Furthermore, there is no single standard method for analyzing the capabilities of a security system.

Security in our world is a race, a race to create obstacles for malicious or inappropriate parties trying to gain access to our private data. This race creates a positive feedback loop illustrated in Figure 3-1. Doing a given analysis indicates what attacks need to be prevented and how destructive they are. To protect against the attacks, the countermeasures come with an intrinsic cost as well as a cost to the asset in terms of accessibility or other factors, requiring further analysis. This simplistic model demonstrates the cycle of security at a high level. This idea is put into practice to develop comprehensive threat analysis methodologies.

An actual structured threat analysis will expand upon these three main components and their interactions. [Teca] and [Tecb] give a basic schema for generating a complete and practical threat model analysis. The more complete and detailed the model, the more likely it will produce useful results. Unfortunately, creating and processing such a model is difficult without a framework.

In this section, we discuss three different methods of threat analysis. The first and simplest is to simply consider threats and countermeasures in an *ad hoc* manner. Second, we examine a system called “practical threat analysis,” which encapsulates the various interactions between attacks, countermeasures, and cost. Finally, we consider case analysis, which puts the *ad hoc* analysis in a context for deeper examination. After examining the benefits of each, we will choose a method for our analysis.

3.2.1 *Ad Hoc* Analysis

Most authors of RFID security papers perform an *ad hoc* security analysis, *e.g.* [BM04, CA06, DWI⁺05, DWIO05, DO05]. The systems are usually examined as an abstract notion, outside the context of their usage. This type of analysis tends to

identify a number of traditional attacks, and the system able to address them while meeting any additional constraints of power, computation, and/or time is considered secure.

The following basic attacks are considered relevant against many RF based communications, including the Identification Friend or Foe systems. Rieback *et al.* puts the RFID attacks in the historical context of such systems [RCT06a].

Sniffing: Most tags are indiscriminate — they will respond to any reader that asks them for identification.

Tracking: By following tag appearances at different readers of known locations, it is possible to determine the tag's location.

Spoofing: An attacker can clone a tag by simply taking the ID he observed during sniffing and writing it to a blank RFID tag.

Replay attacks: An attacker can relay an ID or other signal he sees to another location confusing the service or gaining additional access.

Denial of service: Most commonly, this entails putting an RFID tag where it cannot transmit to a reader.

Unfortunately, considering security to be a simple matter of black and white is shortsighted. For example, it is not reasonable to invest in a multi-million dollar safe to protect a piece of jewelry worth only a few hundred dollars. For effective security, the threat of theft as well as the value of the item need to be taken into account. However, such analysis is problematic in the context of RFID, because the value of the data that could be stolen or modified is unknown. An ID with an expiration date might be associated with a jug of milk or a 50-gallon drum of toxic waste. The monetary value of such data depends heavily upon the individual considering it.

3.2.2 Practical Threat Analysis

[Teca] and [Tecb] describe a more generalized methodology for threat analysis, which takes both the threat of theft and the value of the asset into consideration:

1. **Identify assets:** Determine the financial losses associated with damage to the asset.
2. **Identify vulnerabilities:** Find the potential targets vulnerable to an attack given comprehensive knowledge of the system.
3. **Define countermeasures:** Determine the countermeasures needed to protect against the vulnerabilities and their associated costs.

4. **Build threat scenarios and mitigation plans:** Compose possible threat scenarios and de-construct them into elements and parameters by following these steps:
 - (a) Write a brief description of the scenario.
 - (b) Identify threatened assets and calculate potential damage.
 - (c) Calculate a given threat's probability.
 - (d) Highlight the vulnerabilities exploited by the threat.
 - (e) Create a mitigation plan by picking the most effective countermeasure combination.

The challenge with using this threat analysis system is its dependence on accurate estimates of threat probabilities and asset values. In the area of information management, determining the probability of a given type of exploit is not possible due to the complex nature of software. In addition, the value of data (our assets) may not be easily converted into a quantitative measurement as previously mentioned.

3.2.3 Case Analysis

The disadvantages of *ad hoc* analysis can be corrected by adding context. Knowing what type of data requires protection makes a cursory estimation of value possible. Context will also provide the capability to focus analysis on the attacks possible in a given environment. I will use this modified *ad hoc* analysis for my research. My scenarios and their implications will be described in the next section.

3.3 Scenarios

To better understand the context for my research, let us now consider specific scenarios that demonstrate a need for privacy and security drawn from the military, commercial and medical environments. Through this exercise, the requirements for a well-protected but still practical system will become evident. The areas of attack and relative value of assets will be explored. I make a number of basic assumptions for the purposes of the scenarios:

1. RFID tags and readers are cheap and pervasive.
2. RFID is acceptable for use in most aspects of life.
3. Computing power continues along the current Moore's law trend.

3.3.1 Military Scenarios

RFID's beginnings lie with the military, as mentioned in Section 1.2. It originated as a method for identifying enemy planes and though the technology has improved, many military RFID applications remain of the same nature. Modern planes have very complex IFF systems to figure out who their friends and enemies are with minimal exposure. Due to advances in technology, IFF transponders can now be minimized enough to consider placing them on soldiers.

The supply-chain needs of the military are very similar to those of a consumer or commercial venture: objects need to be reliably transported from one place to another with high visibility¹. Battles have been lost when supplies reached their destinations but became lost. The stakes are high in a military environment, which is often in possession of extremely dangerous items such as anti-aircraft missiles and nuclear weapons. Security is this a primary concern for the military. Conveniently, the resources available to improve security are also usually higher.

The combination of the need for visibility and security have created interesting innovations that will be discussed in the next section.

3.3.1.1 Dog-tags

In the last few years, the military has begun investigating placing RFID transponders built by Irvine Sensors [Irv] inside plastic dog-tags². Dog-tags are the minimal method of identification necessary for soldiers to receive appropriate medical care, supplies, or transport.

RFID-enabled dog-tags have a number of advantages. Significant amounts of stand-alone information can be stored on the tag including blood-type, merits, and commendations. This information would continue to be accessible even if the tag were completely obscured with mud and abrasion, unlike on a bar-code tag. In addition, RFID could be used similarly to the previously mentioned airplane IFF transponders to reduce friendly fire³ [Sha04].

However, RFID-enabled dog-tags also present security challenges. The wearer of the tag would like the information to be easily available in a friendly environment, but the tag should be silent in hostile situations. If the tag was always responsive, a booby-trap type exploit seeking particular RFID transmissions would become feasible. To prevent such exploits, the military needs a secure but environmentally-hardened solution. This means that some of the cost restriction is released, but any failure may have dire consequences.

¹Visibility is the capability to locate an item and any associated information quickly and easily.

²A dog-tag is identification that a soldier wears around his neck in case he is injured or captured. The tags usually contain name, rank, serial number and a religious affiliation. There are actually two tags on the chain. The first tag is only for wearing around the neck. The second tag has a smaller chain for use as a toe tag on a corpse.

³Friendly fire: being shot at by your allies because you've been mistaken for an enemy

An initial test of the electronic dog-tags inspired a novel interface concept of a “silencer” to address the need to toggle the responsiveness of the RFID tag. The standard metal dog-tags make noise when they jingle. However, when a plastic silencer is placed around them, they become quiet. The electronic dog-tags have a protective plastic cover around them and a switch to turn the RFID tag on and off.

3.3.1.2 Military Operation Planning

Another area being explored by the military sector is using RFID networks to assist in planning an operation. Much like in a normal commercial stream, a military operation relies on the delivery of various supplies and equipment to many different places the delivery progress needs to be monitored. The visibility requirement of the resource network provides a good opportunity for deployment of RFID. Just-In-Time logistics for military operations requires having a high level of visibility in the supply chain. Unfortunately, visibility is usually inversely proportional to security. Improving both requires careful design.

The information needs to be secured because supply movements are vulnerable to standard data mining techniques and the knowledge that a particular area is receiving significant supplies signal to the enemy that conflict is imminent. In addition, a terrorist who wants to steal weapons to outfit his faction would like to know where the items are and what route they are taking, so he can steal them at an opportune time.

In the fall of 2003, General Tommy Franks, Commander in Chief of the U.S. Central command, ordered all containers to be RFID tagged at their origin [Pre03].

3.3.1.3 Lessons Learned

We can learn a few important ideas from the military case analysis:

1. Technology user interfaces should be familiar or make use of existing behavior.
2. Users need to have total control over when the tags transmit and when they are quiet.
3. RFID systems should have a form factor similar to the original IDs to minimize retraining.
4. RFID solutions must be hardened for the environment, both physically and electronically.

3.3.2 Commercial Scenarios

The arena most frequently marketed as the place where RFID will greatly speed progress is the commercial sector. It is also most visible to the general public. A

common futuristic example is a cashier-less checkout where a person simply walks out the door of a store carrying items he wishes to purchase. A reader at the door takes care of all the financial transactions from the RFID tags on his credit card and the items.

While the concept of a cashier-less checkout is appealing, it raises concerns about surveillance. How would an individual maintain privacy with a multitude of tagged items on their person?

From a corporate viewpoint, high visibility of the supply chain is both desired and problematic. If a retailer's competitor were able to walk into a factory or store and scan a large subsection of the items housed there, they will have gained extremely valuable information. If the competitor did this multiple times, they could even create a good model of the flow of merchandise through the store. Such information is very valuable for designing the competitor's marketing campaign.

Going even one step further, if the competitor were trying to interfere, they could potentially do so. Assuming rewritable tags [New06], a competitor could easily change the item ID numbers and cause the retailer's inventory system to enter an error state when a shipping manifest did not match. For tags containing additional information⁴, the ability to change it might have further repercussions – replaced source of materials, adjusted expiration dates, or reduced/increased prices. Even worse, the tag's extra capacity might allow virus entry into the system [RCT06b].

Credit card systems are starting to make use of wireless smart cards, and alternative technologies such as Near Field Communications (NFC) [nfc]. A wireless smart card adds close-proximity capabilities to the pre-existing smart card technology. Near Field Communications allow for RFID-like communications with devices such as phones and PDA's, which can consequently be used as virtual credit cards. The RFID tag equivalent in these implementations can access more processing power and memory, making it easier to secure the RF link. Even so, naive implementations make even these technologies vulnerable [Wes06]. An improved design for securing such applications with TagFolio is described in Section 5.1.

3.3.2.1 Lessons Learned

Some important issues brought up by the case analysis within the commercial sector include:

1. Security is important from a liability perspective in financial environments.
2. Each environment and user may have set up very different security exposure levels and a ubiquitous system must be able to adapt.
3. Preventing exploits enabled by tags is not solely the responsibility of the tag designers. Responsibility is distributed system-wide.

⁴This is not the EPC standard, but many manufacturers are pushing for it.

3.3.3 Medical Scenarios

The medical field will see benefit from RFID the fastest. The reasons are straightforward:

1. Mistakes are extremely costly.
2. Equipment, treatments and packaging are expensive.
3. The environment is highly regulated.

Assuming that RFID is heavily implemented, the following scenario that could occur, reveals what problems could arise within the medical field.

3.3.3.1 Bertox

Consider Robert Paulson. Bob is a middle-aged man in his 60's with a physiological problem that requires treatment with medication. Let's call our example medication Bertox; it is produced by Foley Pharmaceuticals to treat the private disease X. The medication bottle has an embedded RFID tag in it.

Bob is concerned about his privacy. He is worried that his insurance company will refuse to pay his claim if he forgets to take his medication. He is worried that his boss Trudy will read the tag on the bottle through his briefcase. He is concerned that the RFID tag might interfere with his ePassport at the airport. He is even slightly worried that one of his neighbors might pick up the tag on one of their readers and associate it with him.

He does not trust that a live RFID tag in his pill bottle is in his best interests while transporting the bottle home, so he decides to use a kill kiosk and destroys the tag. Let's examine the repercussions of Bob's decision. First, Bob's home automation system is now unable to help him take his medicine or find it if he misplaces it.

Let's suppose Bob suddenly has a heart attack from unknown causes. The paramedics rush on-scene and try to use RFID to scan the medications Bob's been taking to examine possible interactions and contraindications. Unfortunately, Bob's tag is dead and the paramedics cannot obtain this information. They notice his MedicAlert tag indicating that he has X, but they don't know that he is taking Bertox, a vasodilator, so they give him nitroglycerine exacerbating Bob's condition.

Bob is rushed to the hospital and unfortunately dies. Alice, Bob's wife, wants to sue Foley Pharmaceuticals as she believes that Bertox was instrumental in his death. The company needs a paper-trail to show that they are not liable, but a lengthy trial would expose too much of their intellectual property. Therefore, they choose to settle out of court. Alice is able to win her case, but Bob is still dead.

Let us now return to the original Bertox pill bottle. The trial is done, the bottle is no longer evidence, and it is time for its disposal. Unfortunately, the deactivated tag means that its proper disposal relies on manual sorting methods. Even though

sorting such an item often takes little effort, Alice can't find the recycling label and instead of recycling it, she chooses to toss it in the normal trash.

If Bob had another option that had allowed his tag to still be accessed while maintaining his privacy, his story have turned out differently. The paramedics might have been able to save him by determining the nature of his medication. Even if Bob had died, Alice might not have bothered with her lawsuit after realizing that the drug was not responsible since the quality control testing results of Bob's lot of Bertox would have been more readily accessible. The bottle would have been properly recycled.

3.3.3.2 Lessons Learned

Bob's story exemplifies the following important points about RFID privacy and security:

1. Entities have specific privacy preferences and feel very strongly about them. Inability to express such preferences and verify that they have been executed makes many users choose a less beneficial route to take.
2. The information associated with a product's development is valuable. Possibly more valuable than what could be gained by its disclosure.
3. Accessibility and accuracy of relevant information in an emergency is critical.
4. The concept of ownership in the electronic domain including RFID is not well defined.
5. Users need to be able to trust and verify that their information will be used to their benefit.
6. Killing a tag improves privacy but at a cost of eliminating many benefits.

3.4 Attacks

As we have been discussing, a number of attacks are possible against RFID systems. In order to design effective countermeasures, we must understand how the attacks themselves function and what situations cause them to arise.

3.4.1 Denial of Service Attacks

A denial of service attack is any attack that completely utilizes a limited resource causing other users to be denied access.

3.4.1.1 Draining a Tag's Power Source

One way to silence an active tag is to query it continuously until its internal power source is empty. This attack works with some semi-passive tags as well, depending upon how much power is used during the signaling process. It does not work at all with passive tags, since the signal itself is the power source.

A simple countermeasure to this type of attack is to deploy the system with mostly passive and semi-passive tags. In an environment where active tags must be used, setting a proper “quench” timeout on the tag reduces the speed at which the reader can run down the tag's power source. Another slightly more far-fetched idea is to build power sources recharge via RF or contacts. Perhaps the limitation of the power source itself should be investigated – tags with long expected lifetimes may need a decaying radioactive source powering a thermoelectric generator⁵.

3.4.1.2 Rogue Blockers

Most literature that describes the positive privacy aspects of blocker tags acknowledges that their inappropriate application is a major detraction. The presence of blocker tags in the wrong place causes all other RFID devices to be obscured. The standard attack using blocker tags is to throw a number of them on multiple shelves, then simply steal the desired item in the confusion.

A countermeasure to this attack is to simply set up a monitoring system and watch for indications of blocker tag presence. This countermeasure invades the privacy of individuals carrying blockers for their localized items.

Juels *et al.* proposed a concept of a friendly or configurable blocker that is able to only mark part of an ID [JRS03]. This “soft blocker” would ask a reader to not read sections of the ID space. As long as the reader complied, the blocker would stay inactive. If the reader attempted to read the forbidden ID space, then the blocker would activate to protect the tags it is associated with.

3.4.1.3 Mass Killing

An attack that many vendors worry about involves unauthorized users entering a tag-dense area and rapidly killing many tags. This situation is called a “mass kill.” While such an attack does not reduce the ability of a legitimate user to access communication mediums, it does reduce access to live tags. For more information on mass kill, refer to Section 6.3.2.

⁵Mind you, there are other complications to this proposal, but it would eliminate the problem of an unreliable power source.

3.4.1.4 RF Jamming

The ether is a shared medium itself. A traditional attack against RF based systems is to produce a very strong signal that overwhelms any other legitimate signal. Such a signal can be generated by a powerful transmitter or a device as simple as a Tesla coil.

3.4.2 Man-in-the-Middle (MITM) Attacks

Man-in-the-Middle (MITM) attacks assume that an attacker is able to listen to traffic between two or more elements in a protocol without changing the data. Active MITM attacks add in the capability for an attacker to stop data from arriving at its destination, modify data, and to send new data.

3.4.2.1 Passive Monitoring and Skimming

The ease of collecting data is both a benefit and a weakness of RFID. In case of promiscuous tags that always transmit whenever asked, placing a reader in a public area such as a mall or airport can gather a significant amount of information. However, information relating to an individual tag is not easily picked out. An attacker who wishes to target a specific individual needs to be able to read the ID on a tag in order to complete a skimming attack.

3.4.2.2 Replay Attacks

Once a tag's ID has been read, it can be retransmitted to emulate the tag in question. For systems that rely on IDs as unique identifiers, such retransmission eliminates any concept of security. Even for systems that do not use an ID for authentication, replay attacks may still be viable. If a challenge and response system does not make use of a random challenge or nonce to verify that an interaction is new, old interactions can be replayed and seem valid.

3.4.2.3 Multi-medium Attacks

Traditional MITM attacks are only able to modify links within a given medium: Ethernet, shared memory, or RF. This includes attacks able to span multiple media or use separate communication channels.

As mentioned in Section 2.1.1.1, attacks capable of determining a tag's presence or secret are possible even when the tag is not actively transmitting. Differential Power Analysis (DPA) [KJJ99] works by examining the signal signature of the tag's powering up. If the tag lacks a regulator or specialized layout geometry designed to resist this attack, the secrets may be exposed.

Another possible attack of this type is Differential Electro Magnetic Analysis (DEMA) [CLRP05], which works by listening to the electromagnetic emanations of a

tag via a separate antenna. By amplifying the noise generated by an operating tag, researchers were able to decode the internal operation of a DESFire card, previously thought to be a secure RFID technology. Such techniques could be applied to other RFID tags.

3.4.3 Social Attacks

One area of attack that should not be forgotten is that of social attacks, where human mistakes or limitations compromise systems that would otherwise be secure. A standard example is a system that requires multiple secure passwords that are often hard to remember. Users who write such passwords down on pieces of paper out of necessity compromise the systems security. Example given in [Sch96] is “rubber hose cryptanalysis”, whereby an attacker recovers a secret key by using physical coercion on the individual trusted with it! Such an attack reminds us of the need for minimizing exposure of sensitive information to the user interface.

An additional social attack considered just as damaging is the failing to change default passwords. Compared to the previous example demonstrating the challenge managing many different passwords, many users fall into the trap of using the same or derived passwords on multiple systems. In some scenarios, we would expect an entire class of tags to have the same access passwords out of convenience. Such a situation creates a tenuous position in security of enforcing such policies without interfering with usability. To address such a possibility, we will need to continue to minimize static secret exposure and develop mechanisms to minimize the damage capability of such an attack.

3.5 System Design

In order for the RFID security system we are designing, TinFoil, to be effective, it must address a variety of functional requirements (FRs) that made clear by the case studies in Section 3.3.

3.5.1 Previous Functional Requirements (FRs)

The RFID system as a whole must be capable of handling a large name-space. The original tag design based on a 96-bit tag comprised $2^{96} = 7.92 \times 10^{28}$ possible IDs. Because of this large number, the following FRs are necessary:

- FR1. The system must be scalable to handle 1 trillion (10^{12}) active IDs. I make an assumption that not all the IDs are active at the same time, as that is beyond any estimation of reasonable usage.
- FR2. The system must continue to work even when the network is segmented or disconnected, though possibly at a degraded capability.

- FR3. The system must minimize bandwidth usage to that of a 56kbit/s modem for home use, and a LAN line for commercial use⁶.
- FR4. The system must leverage existing Internet tools, architectures, and technologies.

Conveniently enough, the above FRs are extremely similar to those used for developing the Domain Name System [Rut97]. This fact allowed us to use DNS to build the initial implementation of ONS that intrinsically meets the requirements. This thesis considers several new FRs for EPC-based systems including ones that incorporate ONS.

3.5.2 Informational Security FRs

The previous FRs in my Master's thesis focused on improving visibility of RFID data. However, total visibility is undesirable in many environments leading to a new set of FRs designed to limit visibility:

- FR5. The system must avoid providing information to unauthorized entities.
- FR6. The system must assist in policy development to define what is and isn't authorized.
- FR7. The system must respect the privacy constraints imposed upon it by society and government regulations⁷.

The leakage of such information in an EPC architecture is shown by the red circles in Figure 3-2. We are not concerned about leakage at the EPC-IS interface because the cryptography and authentication system are being currently developed by EPCglobal.

3.5.3 Human Limitation FRs

The following set of FRs are included because the systems developed in this dissertation are expected to be used worldwide, and by typical human beings:

- FR8. The system must have an intuitive user interface.
- FR9. A description of the system's operation must be translatable to colloquial language.
- FR10. In situations where liability is a concern, such as the medical or legal fields, sufficient protections must fulfill current government standards.

⁶This requirement will change as bandwidth availability increases.

⁷Source routing in the IP protocol was designed to meet this requirement by allowing a requester to designate a desired route around undesirable countries. Unfortunately, various abuses of this protocol caused most installations to disable it [Tan03].

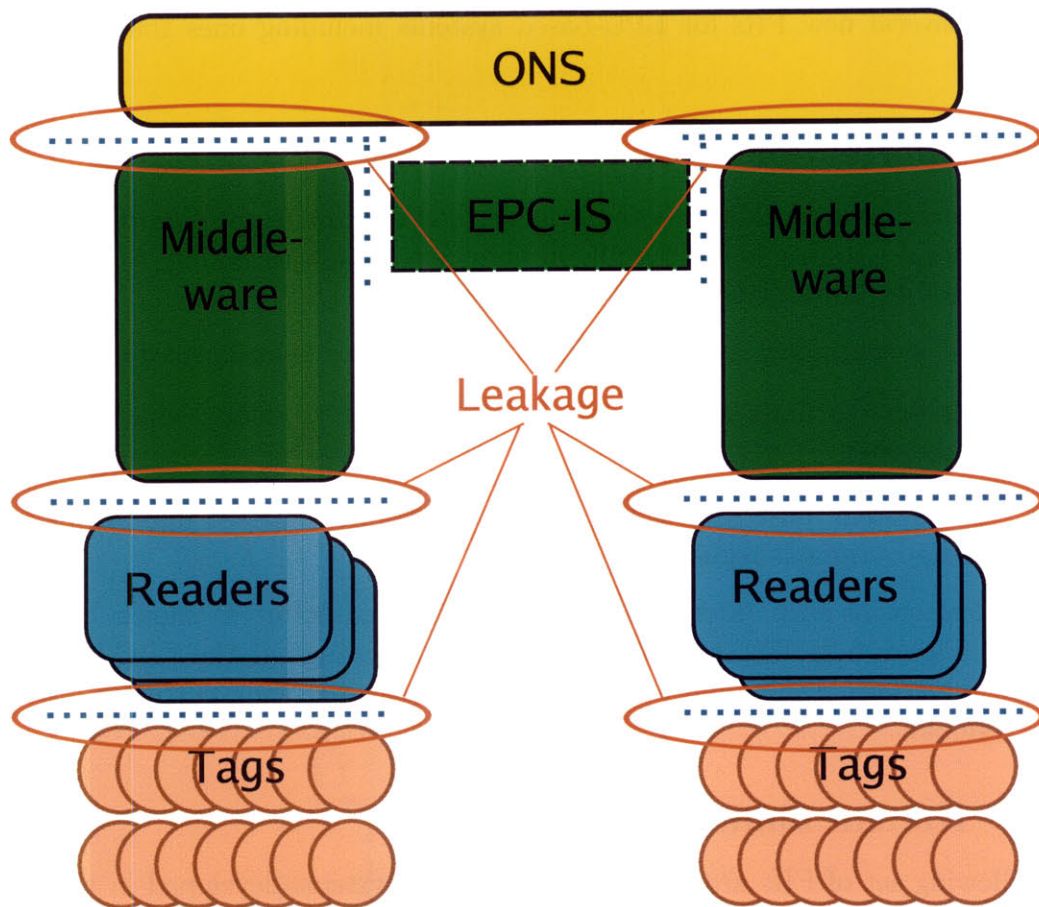


Figure 3-2: Annotated RFID tag life-cycle

3.5.4 Secret Management FRs

As Simson details in [Gar05], discarded data presents a significant problem. Simson reports numerous cases of discarded hard drives being sold or given away still full of confidential data. He explains that these situations arise due to operating system disagreements about the definition of “delete”. Even though the users thought they deleted the data, the operating system did not actually destroy it in a non-recoverable way.

The above issue forces us to stop and think twice before placing secrets on devices in our RFID system. Several tag security mechanisms assume that a pre-existing shared key exists between the reader and the tag. In light of Simson’s discovery, such an arrangement could leak secrets if the device was discarded. To this end, the mechanisms developed as part of my thesis will avoid providing secrets to elements likely to be discarded or repurposed.

FR11. Long term tag secrets must not be stored on readers and personal devices.

FR12. Long term tag secrets must not be exposed to readers and personal devices.

3.6 Choosing a Key or Hash Size

One concern with any secret-based system is the administrative overhead associated with keeping track of a multi-byte secret for each tag. To determine how big a secret must be to protect a given set of sensitive data, we first need to analyze the relative worth and longevity of the data so as not to waste resources. The traditional attacker as mentioned before is trying to counterfeit the identity presented by the tag.

Therefore, as designers, we must create a single design that addresses is capable of coping with ubiquity and a need for high security. When all tags make use of a system’s protection, the amount of data that must be managed becomes quite large. Assuming a lifetime of 100 years and a baseline statistic of 5×10^9 bar-code IDs per year allows us to estimate the number of items as 182×10^{12} .

In addition, based on the 100 year lifetime, we can also approximate the size of the key using Table 3.1 from [Sch96], which gives an analysis of symmetric key-space searches in 1995. Moore’s law indicates that computing power should increase by an order of magnitude every 5 years and thus we can estimate that computers have gotten approximately 100 times faster since 1995. Applying Moore’s law to the 100 year period results in an increase of 20 orders of magnitude of computing power. This seems unlikely, but let us assume this level of speedup is accurate. In order for the key to take 1 year to find after the 100 years have passed, we would need to design a key that takes 10^{20} years to find today. Conveniently enough, the aforementioned table and some extrapolation give us the following estimate of the key size given our

Cost of Hardware	Key size (in bits)					
	40	56	64	80	112	128
\$100K	2s	35h	1y	70,000y	10^{14} y	10^{19} y
\$1M	0.2s	3.5h	37d	7,000y	10^{13} y	10^{18} y
\$10M	0.02s	21m	4d	700y	10^{12} y	10^{17} y
\$100M	2ms	2m	9h	70y	10^{11} y	10^{16} y
\$1G	0.2ms	13s	1h	7y	10^{10} y	10^{15} y
\$10G	0.02ms	1s	5.4m	245d	10^9 y	10^{14} y
\$100G	2 μ s	0.1s	32s	24d	10^8 y	10^{13} y
\$1T	0.2 μ s	0.01s	3s	2.4d	10^7 y	10^{12} y
\$10T	0.02 μ s	1ms	0.3s	6h	10^6 y	10^{11} y

Table 3.1: Average time estimates for a hardware brute-force attack in 1995 from [Sch96] (y=years, m=months, d=days, h=hours, m=minutes, s=seconds)

criteria, with the assumption that a 64-bit key takes 1 year to find in 1995 on \$100K of hardware:

$$64 + \log_2(10^{\frac{100}{5}}) = 130.4 \quad (3.1)$$

Thus, a 131-bit key would be sufficient to last for 100 years against \$100K of hardware, scaled for 100 years in the future. For our purposes, we will settle for a 128-bit key, which will give us 96 years as shown in Equation 3.2:

$$64 + \log_2(10^{\frac{x}{5}}) = 128 \quad (3.2)$$

$$\log_2(10^{\frac{x}{5}}) = 64 \quad (3.3)$$

$$10^{\frac{x}{5}} = 2^{64} \quad (3.4)$$

$$x = 5 \log_{10}(2^{64}) \quad (3.5)$$

$$x \approx 96.3 \quad (3.6)$$

Assuming we are using the 96-bit EPC tags, each with a 128-bit key in separate storage, this comes out to a staggering 5,096 Terabytes of data (Equation 3.7) that needs to be distributed.

$$\frac{1}{8}(96 + 128)(182 * 10^{12}) = 5.096 \times 10^{15} \quad (3.7)$$

One reassuring fact is that data storage and bandwidth are also increasing at a tremendous rate, making this less of an issue.

3.6.1 An Issue with EPC Gen2

EPC Gen2 uses two 32-bit secrets for access and kill authentication [EPC05a]. 32 bits gives $2^{32} = 4.29 \times 10^9$ possible keys to search through. This number seems large, but modern computers can easily mount a dictionary attack⁸ against such a key-space.

Considering the Gen2 specification, we have a couple of possibilities for improving security:

1. Consider chaining the passwords together to make one 64-bit password, creating a much safer search space with $2^{64} = 1.845 \times 10^{19}$ possibilities.
2. Expand the tag size to accommodate a larger key-size.

One of the goals of my research is to minimize cost, so option 1 seems more attractive. In order to encode the disposition of such a secret, we place the original secret as the first 32 bits.

The 64-bit kill secret would then appear as:

KILL [32] ACCESS [32]

The 64-bit access secret would be:

ACCESS [32] KILL [32]

This is still insufficient to gain the full strength of a 128-bit key as we would desire, but it increases security by vastly increasing the numeric search space. A detraction of this scheme is that it couples the two secrets together. You cannot provide only ACCESS privileges – you must give both secrets to perform either type of access.

3.7 Lifecycle

We can analyze an RFID system in one of two ways:

1. as a time-independent set of interactions between elements
2. as a lifecycle of the tag

Previous compositional designs chose to model RFID systems using the first concept based solely on element interactions. However, considering the lifecycle of the critical tag element is more intuitive. A tag's lifecycle is composed of the following stages (illustrated in Figure 3-3):

1. Born:

⁸Dictionary attack: pre-computing all possible keys with a known plain-text to determine which key was used

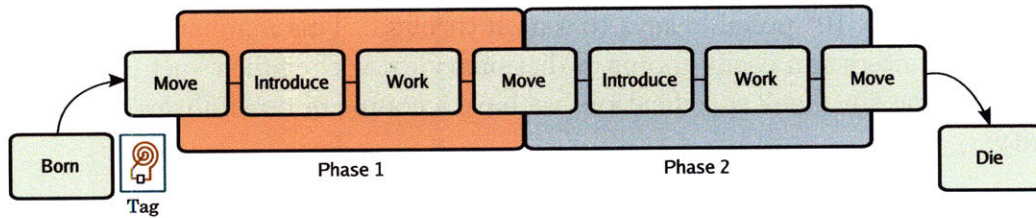


Figure 3-3: Typical RFID tag lifecycle

- (a) Tags are born and given identity.
 - (b) Tags authenticate their identity to a reader.
2. **Move:** Tags move to another owner and/or location.
 3. **Introduce:** Tags are introduced to a reader, which checks their authenticity.
 4. **Work:**
 - (a) Tag information is used to resolve their purpose via ONS.
 - (b) Tag updates allow them to be tracked.
 - (c) Tags are given commands, often to change state.
 - (d) Tags have their secrets and passwords updated.
 5. **Die:** Tags are given kill commands to deactivate them.

This thesis groups elements into the major interactions during a tag's lifecycle (listed below). The manufacturing operation of producing a tag (**Born**) is beyond the scope of my thesis.

Identity and introduction: JanusTag to hide a tag's identity and TagCheck to prove its authenticity

Work: TorONS to anonymize ONS look-ups

Move: TagFolio, which allows transfer of ownership without revealing a tag's secrets to the intermediary

Die: TagDirective provides time-limited access control, Exponential Rampup makes mass kills difficult, and Neighborhood Watch provides a communication channel for protective readers to determine which kill operations are authorized

3.8 Summary

In this chapter we have discussed the various issues that drive the development of our desired RFID design. These issues have guided the design of TinFoil by making clear the FRs that need to be satisfied. These FRs, in turn, have provided the necessary framework and specifications for the compositional elements to construct our system.

In the next chapter, we begin our exploration of tag identity and authentication. This chapter will provide a technological foundation for other innovations to build upon.

Chapter 4

Tag Identity and Verification

“The first group I went to, there were introductions: this is Alice, this is Brenda, this is Dover. Everyone smiles with that invisible gun to their head.

I never give my real name at support groups”[Pal96]

An inherent problem with ID systems is verification of tag and item proximity to each other. Someone listening to a tag could theoretically mimic its response elsewhere and essentially “become” the tag/item. Distance bounding protocols [HmGK05] and ultra-wide-band modulation [YSH06] attempt to ensure that tag proximity to prevent a relay attack. However, neither protocol verifies the authenticity of the tag/item combination.

This problem is especially relevant in situations where value is associated with an authentic item. The average consumer would be slightly inconvenienced if the batteries he bought turned out to be counterfeit rather than genuine Duracell. The Duracell corporation, on the other hand, would suffer significant financial losses if the market were flooded with counterfeits of their product. As increased value items such as \$100 poker chips begin to use RFID for verification [GPJ04], the desire to make illicit copies will also rise.

To illustrate the same point with yet another example, let’s consider SpeedPass¹ transponders used to make toll taking easier and much faster. SpeedPass includes an RFID tag made by Texas Instruments called a Digital Signal Transponder (DST). In February of 2005, a team of researchers succeeded in reverse engineering the DST protocol and showed its extreme insecurity [Jue05a]. This might seem like a minor point since the tags are usually used for taking tolls out of a bank account, but consider the repercussions of such an insecure system. If the attack were easy to repeat, one could argue in a court of law that the large toll fee charged to their account was not for them. In addition, a new black market with copied SpeedPasses

¹*a.k.a.* Fastlane in Massachusetts, and EZ-Pass in the rest of New England.

may arise. Whenever financial transactions take place using insecure technology, the consumer should worry.

Many of these cryptographic protocol problems stem from the choice to develop cryptographic protocols in a proprietary manner, thus bypassing the valuable peer review process. This was seen in the case of the DVD content scrambling system (DVDCSS) [BRSG06] and the GSM security framework [GSM]. Both of these cryptographic systems failed to follow a basic premise first described by Dutchman A. Kerckhoffs in the nineteenth century:

“Secrecy must reside entirely on the key, and nothing else.” [Sch96].

Our goal must be to avoid making the same mistake.

Using basic cryptographic elements, we can create a solution in the form of a very simple authentication protocol called TagCheck² [Fol03b]. In this protocol, a reader accesses a database and a secret number on the tag, and generates a challenge-response pair that is compared. This protocol can also be used to develop a more secure kill system.

The important design constraint TagCheck is that the reader never knows the actual secret key on the tag, and thus, if compromised, it cannot reveal the secret information. Furthermore, TagCheck implementation requires only a minimum of additional electronics including a good but small-footprint hash function on the tag (as proposed in [Wei03]) and control software on the reader.

4.1 Two-Faced Tags: JanusTag

A persistent problem with many RFID systems is that IDs, whose identity stays the same, are easily traceable. To solve this problem, some systems propose rewriting the tag’s identity each time the tag is moved to a new environment [MW04]. However, this solution fails if the mapping database becomes corrupted. I propose a simpler solution: give a tag two identities, each with a different purpose. Much like Stevenson’s Jekyll and Hyde [Ste08], the two identities each have a specific purpose – an ephemeral public identity is shown to unauthenticated readers, while a static private identity is only visible to authenticated readers.

The proposed tag’s two identities are Public ID (PID) and Real ID (RID). A PID is given to any interrogator who asks, unless the tag has been silenced or killed. The RID is only shown to interrogators who have authenticated to the tag via other means. The RID might also be exposed when the tag is placed in the UNLOCKED mode.

Tags should have their PID set to RID when transferred to a customer. The customer then has the option to disguise the tag via a separate authentication mechanism. Once authentication is complete, PID is set to a separate identity. In the

²previously referred to as TagSeal (also its name in our patent

future, the customer can change the PID of the tag if they suspect a tracking attempt³. Additionally, a separate mode called CLOAKING would listen for other tags responding in an field and choose one of the identities to use for a period of time. A more stochastic approach would be to have a CHAFF mode where the ID changes its identity randomly. Such changing ID's would make tracking extremely difficult.

To further obscure the IDs, you could also deploy the Reader Misdirection mechanism described in Section 6.2.1. Doing so could be used to allow the PID fake identity to be sustained while the PID changes and continues along a different course.

RID can be Read Only (RO) or One Time Programmable (OTP), reducing its complexity and cost. The PID, on the other hand, needs to be rewritable. This would put our design in the EPC Class 2 specification.

The concept of JanusTag will be used heavily in Section 5.1, though it is a stand-alone mechanism that can be added to most protocols to enhance privacy with minimal effort.

4.2 Introduction to Anti-Counterfeiting

Many current uses of RFID for counterfeit protection rely on the existence of RFID tags and readers outside of the public domain. This will prove false, however, since the underlying design that makes tags and readers useful requires ubiquity. Let's examine the basic anti-counterfeiting protocol to expose some of its vulnerabilities. The steps in the basic protocol follow a pattern such as this:

READER
Are there any tags out there?

REAL TAG
I'm tag 31337, trust me.

READER
You're 31337? Neat!

The above protocol is very vulnerable to a trivial attack. An attacker simply needs to listen in when the reader acknowledges the tag, proceed to make their own copy of the tag. The fake tag can then respond to the reader the next time it asks:

READER
Are there any tags out there?

FAKE TAG
I'm tag 31337, trust me.

³a.k.a. "ditching the tail"

READER

You're 31337? Neat!

In order to improve security, we would prefer for the above interaction to proceed in the following way:

READER

Are there any tags out there?

FAKE TAG

I'm tag 31337, trust me.

READER

Really? What is 31337's favorite color?

FAKE TAG

Ummmm. Blue?

READER

Sorry. Call security!

Even this interaction is insecure, though, because an attacker can listen in when the real tag communicates its favorite color (*i.e.* its password) to the reader. He can then swap the fake tag in and repeat the password:

READER

Are there any tags out there?

REAL TAG

I'm tag 31337, trust me.

READER

Really? What is 31337's favorite color?

REAL TAG

Pantone 142.

READER

That's right!

(Attacker swaps the two tags.)

READER

Are there any tags out there?

FAKE TAG

I'm tag 31337, trust me.

READER

Really? What is 31337's favorite color?

FAKE TAG

Pantone 142.

READER

That's right!

To improve security even more, a Zero-Knowledge Proof (ZKP) system such as that described in Appendix A.5 would be preferable. In a ZKP system, one entity proves the existence of a secret to another without revealing the secret. Unfortunately, zero-knowledge proofs are very computationally intensive. Instead, we need a simple challenge-response protocol that the tag and the reader can use to achieve the same result. An example of such a challenge-response interaction is illustrated here:

READER

Are there any tags out there?

REAL TAG

I'm tag 31337, trust me.

READER

Really? What would be the first thing to come into 31337's head if I said "My Hovercraft is full of eels"?

REAL TAG

The eagle has landed.

READER

That's right!

(Attacker swaps the two tags.)

READER

Are there any tags out there?

FAKE TAG

I'm tag 31337, trust me.

READER

Really? What would be the first thing to come into 31337's head if I said "The football is in play"?

FAKE TAG

Wait! You should have asked about the Hovercraft!

READER

Sorry. Call security!

In this whimsical example, an attacker could theoretically watch many transactions to determine what the right answer to a given challenge is. This attack stresses the importance of choosing random non-repeating challenges and an algorithm that does not produce predictable results to someone not privy to the shared secret.

The protocol between the reader and the tag must be light-weight and computationally non-intensive. This is because to reduce cost and increase range, the tag has very little power (energy and computation) available to it. Saving state is expensive, requiring additional memory that can be written to. Our design is expected to be applied EPC Class 1 tags which have very limited capabilities. The protocol between the reader and other servers can be heavier, however, as it has less restrictive power requirements.

4.2.1 Reader-to-Server Assumptions

Modern RFID readers come with modern computer capabilities: networking, memory, and general purpose CPU's. Given these features, there are a number of mechanisms that could be used for authenticating one reader to another. A fairly simple one involves having a centralized authority sign a certificate for each of the readers. The readers can then use an SSL/TLS authentication [TLS] and encryption mechanism for any communication⁴. The challenge with this approach is the same as for any other public-key cryptography, namely key distribution. However, since readers are physical electronic devices, they can use a trusted channel such as a wire protocol. Other mechanisms similar to those discussed in Section 2.8 are also available for key predistribution.

For the purposes of my work, I make an assumption that one of these reader to server authentication mechanisms is already in place. Specifically, I assume that the reader $\mathcal{R}$ and any servers it needs to access have set up a secure two-way connection using Kerberos 5 or SSL/TLS. Lighter-weight protocols would be preferable, but since there are sufficient existing mechanisms, I will assume the system is functioning.

⁴more about SSL/TLS in Appendix A.4

4.3 Hash-Based TagCheck

I will now describe TagCheck, a protocol I designed for authenticating a tag to a reader. This protocol requires some cryptographic primitives, but with a minimum of modifications and additional cost. I will start by giving an overview of the protocol, then explore the challenges of implementing it while maintaining a low-power low-cost design.

4.3.1 Design

There are additional assumptions we must make about the problem of RFID tag verification in order to construct a sensible solution. These are:

1. There is enough processing power between $\mathcal{R}$ and $\mathcal{V}$ to set up a secure connection as discussed in Section 4.2.1.
2. An attacker will not be spoofing on both the RF and the physical network channel. If an attacker can spoof on both channels, he will be able to convince the reader of false authenticity regardless of how the protocol is constricted. This attack is difficult if the first assumption is true.
3. There is very little processing power and state on the tag $\mathcal{T}$. Moore's law is pushing this boundary, but it does still hold today.
4. There exists (and we can use) a one-way collision-resistant hash function or block encryption function that can fit on $\mathcal{T}$ given its size and power constraints. We can use the design given in the previous section or one of the other schemes.

When $\mathcal{T}$ is programmed, it is also given a secret (s) in addition to its ID i . s is also stored in the TagCheck verification server $\mathcal{V}$. With similarities to Fiestel's 2-pass authentication system [FNS75], we can use hash functions or symmetric encryption to generate our authentication mechanism. Traditionally, hash functions have been considered much slower and more expensive to implement than an equivalent block cipher function. In Chapter 4.4.3, we will show that a hash function can be synthesized using block ciphers, making the two solutions nearly equivalent in implementation cost.

4.3.2 Protocol

When $\mathcal{R}$ wishes to verify $\mathcal{T}$'s authenticity, it completes the following steps (also illustrated in Figure 4-1):

1. $\mathcal{R}$ queries $\mathcal{T}$ for its ID.
2. $\mathcal{T}$ responds with i .

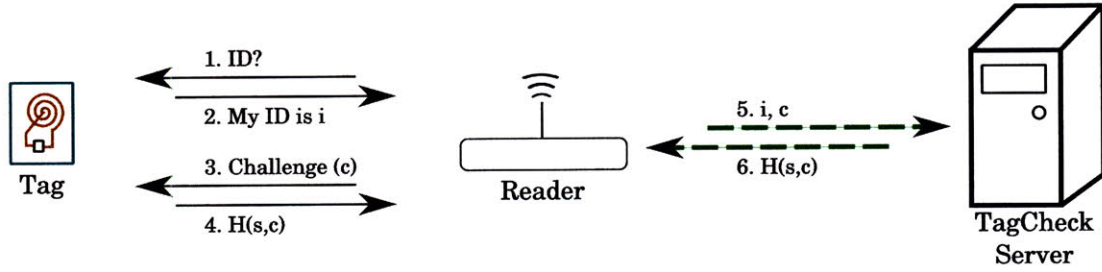


Figure 4-1: Hash-based TagCheck operation

3. $\mathcal{R}$ creates a random number challenge c and sends it to $\mathcal{T}$.
4. $\mathcal{T}$ creates a hash function $H(s,c)_1$ and returns it to $\mathcal{R}$.
5. $\mathcal{R}$ sends c to $\mathcal{V}$ via the secure link.
6. $\mathcal{V}$ creates $H(s,c)_2$ and returns it to $\mathcal{R}$ via the secure link.
7. $\mathcal{R}$ compares $H(s,c)_1$ and $H(s,c)_2$ for verification.

Details of the hash function $H()$ synthesis are explained in Section 4.4.3.

4.3.3 Hash-Based TagCheck Advantages

The elegance of the TagCheck design is that the reader is never entrusted with any secret data, but it is nevertheless able to verify the secret on the tag. Furthermore, the secret is never transmitted in a manner that can be reversed thanks to the one-way hash function. The only information stored on the reader is its credentials for the secure connection to the verification server.

A significant advantage that TagCheck has over other similar systems comes from the reader's responsibility for both ends of the verification step. At first glance, this may seem like a disadvantage. However, by decoupling the tag-reader protocol from the reader-server link, the reader is now free to do bulk and offline verification at a later time.

Let us illustrate the above advantage with an example. Suppose you are at a bazaar and wish to check the authenticity of several RFID tagged items. To check the items with an on-line interactive tool, you'd need to have a constant Internet connection on your hand-held reader which might not be possible. However, with TagCheck, you can challenge a number of tags, then verify the results when connectivity is available, *i.e.* an Internet kiosk.

In addition to the above, bulk verification and bulk offline verification of many tags also become more efficient. Instead of having to wait for response from a server

for each tag, the reader can scan all the tags, then queue them up for verification during idle time.

If we could assume that an attacker would not attack the $\mathcal{T} \leftrightarrow \mathcal{R}$ and $\mathcal{R} \leftrightarrow \mathcal{V}$ links simultaneously, we wouldn't even need the secure link for $\mathcal{R} \leftrightarrow \mathcal{V}$. However, a determined attacker might be willing to do so and we must therefore take precautions. If we left the $\mathcal{R} \leftrightarrow \mathcal{V}$ connection insecure, an attacker could convince $\mathcal{R}$ of $\mathcal{T}$ false authenticity by giving a bogus response as the response to the challenge and using the same response again during the validation step. To protect against both channels in the TagCheck protocol would require more computation on the part of the tag, which seems unlikely. Instead, it is the responsibility of the reader to have a secure connection to the verification server.

One concern is that repeated challenges might leak information to a monitoring agent or be vulnerable to a replay attack. An attacker who sees the random number challenge for an exchange will only be able to fake the response to that given random number. Assuming a good generator such as [FS03b], the chances of that are small.

4.4 Hash Functions on RFID

Creating a cryptographically sound hash function on an RFID tag is a non-trivial task, as mentioned in [Wei03]. Deploying RFID tags in a normal environment is difficult and the challenges only grow when we ask the tags to perform computationally intensive tasks such as hash function generation. Some such challenges include:

1. Power is not guaranteed for any length of time.
2. Transmissions may end at any moment without warning.
3. Transmissions may be one-sided, especially since the transmission range of the tag is significantly shorter than that of the reader.
4. Saving state requires a significant amount of power and additional silicon memory.

4.4.1 Hash Functions Explored by Weis

In his Master's thesis, Weis explored many different minimal-implementation hash function designs [Wei03]. He suggested the most promising areas as cellular automata and nonlinear feedback registers; both can be implemented in very small numbers of transistors.

Cellular automata hash functions make use of the chaotic nature of certain types of cellular automata simulations. A hash can be produced by using the message values as initialization parameters for the simulation. Unfortunately, these hash functions, including the described CellHash [MZI98], are believed to intrinsically need large

numbers of cycles to complete a single hashing operation. Additionally, it is believed that these operations cannot be optimized for parallel processing. The large number of computations translates into significant computation time, a problem for the RFID timing constraints.

Nonlinear Feedback Shift Registers (NLFSR) have a similar challenge of designing a small but effective feedback function that will exhibit the properties of confusion and diffusion when applied repetitively on the shift register. These NLFSR are building blocks of many symmetric key ciphers, including the Fiestel cipher DES. Weis describes a possible implementation of a secure NLFSR function that needs a 256-bit shift register [Wei03]. This hash function would still require a large number of processing rounds and a the large shift register, but implementation of such a device is within reason. The construction of a provably secure NLFSR is open research.

Given the state of current research, it seems likely that a sufficiently small-footprint hash function for RFID applications will exist soon. Our design is not specific to any particular cryptography hash function – any implementation that satisfies the security constraints⁵ placed on it by the RFID environment will be sufficient. Alas, we cannot wait for such developments, so we will construct a suitable hash function using current technology.

4.4.2 Controlled Physical Random Functions (CPUFs)

[GCvD⁺02a] proposes a Controlled Physical Random Function (CPUF)⁶, a concept useful to generating a hash function with a minimum number of gates. A CPUF has a controlled delay feedback loop, which stabilizes the output of a function and uses minute differences in manufacturing. The system generates a number associated with a chip that is randomly distributed and tamper-proof⁷. With some minor modifications, this mechanism can be used to generate a hash function using very few transistors.

Unfortunately, the manufacturer has no control over what ID or hash the chip will be. Furthermore, the number cannot be changed later, as it is effectively a fingerprint of the chip. In the same vein, the hash function cannot be easily characterized. Instead, current proposals build a table of known valid inputs and outputs to be used later. This has problems for when significant numbers of accesses need to be made, presenting the possibility of a denial of service attack.

In its pure form, the CPUF is not very useful for our application. A circuit that cannot be copied is not enough to prove authenticity of a tag. A malicious attacker could skim the ID and program another device to emulate that ID.

⁵timing and hash code size

⁶The acronym actually stands for Controlled Physical Unclonable Function. The alternative acronym was created to avoid collision with Pseudo-Random Functions.

⁷If the designer routes the delay-wires through the middle of the chip, or on a top layer, anyone trying to modify the chip will scramble the number.

4.4.3 PGV Hash Function Synthesis

Another way to create a cryptographically secure hash function is to synthesize it from a block cipher. The method for doing so is well known [PGV93], but I have not found in the literature any instances suitable for RFID applications which I will develop in this section. Before I do, let's review the synthesis method designed by Preneel *et al.* [PGV93]. In this work, Preneel *et al.* describes 64 different methods for generating a synthesized hash function, categorizing the strongest attacks against each scheme. These attacks are described in detail in Section 4.4.4. Preneel *et al.* conclude that only 4 of their schemes are secure and 8 schemes are only vulnerable to a weak attack known as a fixed point attack. Black *et al.* further analyzed Preneel *et al.*'s methods and discovered that an additional 8 of the 13 schemes thought to be weak against a backward attack are actually similarly collision resistant⁸ as the 12 strong methods [BRS02]. These so called group-2 schemes (as [BRS02] names them) are only weaker in their inversion resistance. We will now follow the method of Preneel *et al.* to synthesize our secure hash function using AES using the notation given in Table 4.1.

X	plain-text
K	key
C	cipher-text
$C = E(K, X)$	block encryption of X by K
$X = D(K, C)$	block decryption of C by K
n	block size in bits
k	key size in bits
t	number of blocks of n in X ($t = \frac{X}{n}$)
X_i	i th block within X
$f()$	round function
H_0	initial value (IV)
H_{t-1}	hash code from previous round
H_t	hash code, <i>a.k.a.</i> the output
R	ratio of encryptions per processing of n bits
,	indicates a different, but similar value or function

Table 4.1: Notation used in hash function analysis

The synthesized hash has the general form of:

$$H_i = f(X_i, H_{i-1}) \quad i = 1, 2, \dots, t \quad (4.1)$$

⁸resisting attempts to find inputs producing the same output.

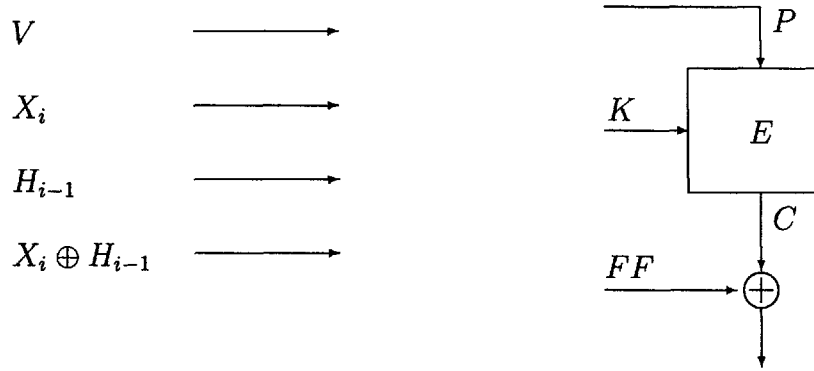


Figure 4-2: General layout of a synthesized hash function [PGV93]

The majority of the work in [PGV93] focuses on the description, categorization, and analysis of the round function. To make the synthesis and analysis simpler, the authors assume $k = n$. We will refer to this method as “PGV” for consistency. As shown in Figure 4-2, a block cipher has 2 inputs (key input $\mathcal{K}$ and plaintext input $\mathcal{P}$)⁹ and one output (cipher-text $\mathcal{C}$). An XOR operation $\oplus$ of Feed Forward ($\mathcal{FF}$) the output of $\mathcal{C}$ is our round output. Each of these inputs can take one of four possible outputs: V , X_i , H_{i-1} , or $X_i \oplus H_{i-1}$. This gives a total of 64 (4^3) possible combinations.

4.4.4 Attacks against Hashes

[PGV93] provides a good taxonomy of attacks (summarized below) for the various combinations of the round function $f(X_i, H_{i-1})$. Before we consider the attacks, we must understand the goals of an attacker: collision and function inversion. A collision occurs when two input values return the same output value. Collision resistance indicates the difficulty of finding collisions [FS03a]. When a hash function’s inputs can be determined from its output, then it is vulnerable to function inversion. Functions that cannot be inverted are called one-way functions.

Direct attack (D): Given H_{i-1} and H_i , it is reasonable to find X_i .

Permutation attack (P): Consider:

$$H_i = H_{i-1} \oplus f'(X_i)$$

where $f'()$ is an arbitrary one-way function. The hash code is independent of the order of the message blocks in hashes vulnerable to this type of attack, easing the collision search. Block replay is also viable.

⁹The plaintext input $\mathcal{P}$ is chosen to be easy to differentiate from the actual plaintext X .

Backward attack (B): Using H_i , finding (X_i, H_{i-1}) that satisfies $f(X_i, H_{i-1})$ is easy. Finding a preimage or second preimage with a random IV is trivial. Meet-in-the-middle attacks¹⁰ are valid for finding a preimage or second preimage.

Forward attack (F): Given H_{i-1} , H'_{i-1} , and X_i when H_i is fixed, the attacker can construct a X'_i such that:

$$f(X'_i, H'_{i-1}) = f(X_i, H_{i-1}) = H_i$$

If this attack is valid, generating a second preimage for a given hash code is easy. It is not necessarily easy to make a second preimage for a given element in the range.

Fixed point attack (FP): A viable attack is to look for H_{i-1} and X_i that satisfy $f(X_i, H_{i-1}) = H_{i-1}$. This weakness is considered minor because in a one-way function situation, it is very unlikely that the desired message will match the located H_{i-1} .

4.4.5 Selection of PGV Function

ID Number in [PGV93]	Function Expression
1	$E(H_{i-1}, X_i) \oplus X_i$
2	$E(H_{i-1}, X_i \oplus H_{i-1}) \oplus X_i \oplus H_{i-1}$
3	$E(H_{i-1}, X_i) \oplus X_i \oplus H_{i-1}$
4	$E(H_{i-1}, X_i \oplus H_{i-1}) \oplus X_i$

Table 4.2: Most secure synthesized hash functions [PGV93]

The 4 schemes that were the strongest are described in Table 4.2. The collision and inversion resistance bounds determined in [BRS02] are the same for all 4 functions and are given below:

¹⁰A meet in the middle attack decreases the search time for a key by encrypting a known plaintext with all possible keys and decrypting a known ciphertext with all possible keys to find the individual round keys in a chained encryption system.

$$\begin{aligned}
\text{Collision Resistance Low Bound} &= 0.039 \frac{(q-1)(q-3)}{2^n} \\
\text{Collision Resistance High Bound} &= \frac{q(q+1)}{2^n} \\
\text{Inversion Resistance Low Bound} &= \frac{0.4}{2^n} \\
\text{Inversion Resistance High Bound} &= \frac{2q}{2^n}
\end{aligned}$$

q : the number of random oracle queries made for an attack
 n : total number of queries

Any of these 4 PGV combinations would be sufficient for a cryptographically strong hash function, so we choose a scheme requiring the fewest additional gates, $E(H_{i-1}, X_i) \oplus X_i$. We will examine applying Feldhofer *et al.*'s implementation of Advanced Encryption Standard (AES)¹¹. This implementation comprises 13,600 transistors and uses 8 micro amps of power at 100 kHz¹² [FR06]. A completed AES block cipher takes 1,032 cycles, resulting in a period of 0.01 second per encoding at 100 kHz. This is sufficient for reading a fairly large number of tags and still be unnoticeable at the human interface level. This is too slow to meet the requirements of the EPC Gen2 standard in its current state. Changes to allow slow algorithms such as AES to meet these timing requirements are mentioned in Section 4.6.

Another option for a suitable block cipher with a reasonable layout size is DESL, a version of the Digital Encryption Standard (DES) redesigned for use with RFID by Poschmann *et al.* [PLSP06]. However, DESL has a block size of 64 bits and a 56-bit key size, which would only produce a 64-bit hash, insufficient for our needs. We prefer the 128-bit keys that AES uses, and the 128-bit output of a hash function based upon AES. This is half the size recommended by Ferguson and Schneier [FS03a], but we are limited by the block functions available. A 128-bit key size gives an expected utility of 96 years as discussed in Section 3.6.

An implementation for generating a cryptographic hash function using AES is outlined in Figure 4-3. For a 128-bit output, the hash function is operated in the following way:

1. $n = 0$
2. Initial value (IV) and first message block m_0 are loaded into RAM.
3. IV is sent to the AES key K input in 8-bit chunks.
4. m_n is sent to the AES message M input in 8-bit chunks.

¹¹refer to Chapter 2.2.2

¹²A UHF Gen2 tag needs to operate at 1.92MHz or higher to meet the timing constraints [Cor].

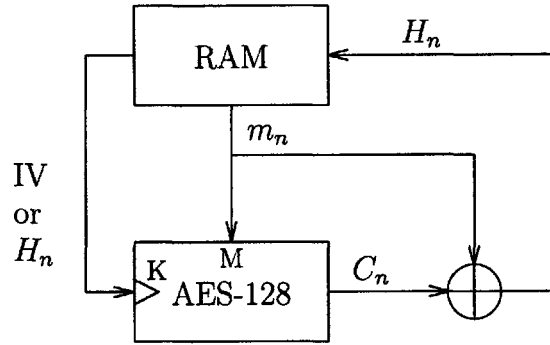


Figure 4-3: Block diagram of an AES hash function

5. $C_n = E_{IV}(m_n)$.
6. $H_{n+1} = m_n \oplus C_n$
7. H_{n+1} is stored in RAM.
8. $n = n + 1$
9. H_n is sent to K.
10. m_n is sent to M in 8-bit chunks.
11. $C_n = E_{H_n}(m_n)$
12. $H_{n+1} = m_n \oplus C_n$
13. The result is sent to the RAM and stored in a temporary location.
14. Steps 8–13 are repeated until n is the # of last block.
15. The value stored in the temporary location H_n is the 128-bit hash.

The EPC Gen2 specification mandates that the various secrets (access and kill passwords) must be 32 bits in length. This is one quarter of the size of an AES key or block path, which are 128 bits¹³. To use the implementation described in Figure 4-3, we use random values or any other type of padding. Conveniently, the output of the synthesized hash function is 128 bits, which is 4 times as large as a Gen2 password. Proper interface is important in implementation details.

The AES implementation given by [FR06] takes 8-bit inputs for the key and block, and will output the result in 8-bit chunks. This is convenient, as the only additional needed element for synthesizing the hash is an XOR gate. An 8-bit 2-input XOR gate

¹³AES has a block path of 128 bits and a key size of 128, 192, or 256 bits. Rijndael, which AES is a specific instantiation of, can support any key or block size in multiples of 32 bits.

is only 8 gate equivalents or 12 transistors. The total transistor count for an AES-hash implementation is given in Table 4.3. We will also need a few sets of 128 bits of RAM in order to hold the initial value, message, and result. Using the low-power SRAM design from [Mar99] as a baseline, we need 7 transistors per bit for memory.

Module	Number Needed	Total Gates	Total Transistors
AES	1	3,400	13,600
XOR	1	8	12
Key Memory	128	N/A	896
Block Memory	128	N/A	896
Temp Memory	128	N/A	896
Grand Total			16,300

Table 4.3: Gate and transistor counts for AES hash implementation

4.4.5.1 Random Number Generation: Fortuna

The AES implementation also allows us to use a simple and powerful pseudo-random number generator called Fortuna developed by Ferguson and Schneier [FS03b]. Fortuna generates a random number in a similar manner to how our hash function is synthesized.

Here is a high-level overview of the Fortuna operation:

1. The key K and counter C are set to zero.
2. $C = C + 1$
3. r is initialized to an empty string.
4. Reseeding occurs: $K = H(K||s)$ where s is a new seed.
5. $r = r||E(K, C)$
6. $C = C + 1$
7. Occasionally, reseeding at step 4 is repeated.
8. Step 5 is repeated.

Making use of Fortuna has the same basic requirements as the AES hash.

This AES hash function allows the hash-based TagCheck protocol to be implemented on high-end RFID tags. In the event that our AES hash design is determined to be unsuitable for such applications, we can also design a version that uses symmetric cryptography.

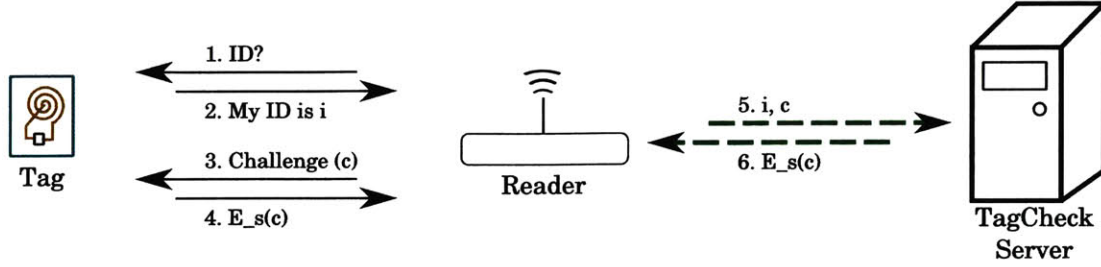


Figure 4-4: Symmetric cipher TagCheck operation

4.5 Symmetric Cipher TagCheck Protocol

The initial design of TagCheck focused on the capabilities that a hash could provide. We subsequently realized that the same concepts could be applied to a modified version of the Fiestel 2-pass authentication. This new protocol is detailed in Figure 4-4) and follows the below steps:

1. $\mathcal{R}$ queries $\mathcal{T}$ for its ID.
2. $\mathcal{T}$ responds with i .
3. $\mathcal{R}$ creates a random number challenge (c) and sends it to $\mathcal{T}$.
4. $\mathcal{T}$ creates $E_s(c)_1$ and returns it to $\mathcal{R}$.
5. $\mathcal{R}$ sends c and i to $\mathcal{V}$ via the secure link.
6. $\mathcal{V}$ creates $E_s(c)_2$ and returns it to $\mathcal{R}$ via the secure link.
7. $\mathcal{R}$ compares $E_s(c)_1$ and $E_s(c)_2$ for verification.

4.5.1 Advantages

An obvious advantage to this design is the lesser requirement of a block cipher. This reduces the transistor count and should make the protocol operate significantly faster. To examine why speed is particularly important for RFID authentication, we will examine RFID timing parameters in the next section.

4.6 Timing Constraints

The key to interoperability for any system is standards. The two standards that are the most prevalent in RFID are the ISO/IEC 18000 standard and the EPCglobal tag standards. In this section, we will discuss what limitations these standards place on

RFID tags, and how we can work around these limitations by increasing processor speed or using a concept of interleaving.

4.6.1 ISO/IEC 18000

Let us now consider the timing parameters of our AES hash implementation in the context of international standards. Tags must have a clock speed of at least 1.92MHz in order to meet the Gen2 specification [Cor]. The ISO/IEC 18000 standard for 900 MHz tags has a maximum delay d of 320 μ s between a reader's query and a tag's reply. For the 1.92MHz frequency f , this only provides 614 operations o .

$$o = d \times f \quad (4.2)$$

$$= 320\mu\text{s} \times 1.92\text{MHz} = 614.4 \quad (4.3)$$

Unfortunately each AES cycle takes 1,032 clock cycles. Each run through the AES hash takes 2 AES cycles and additional register loading time, resulting in a total of 2,096 cycles:

$$(1032 * 2) + (16 * 2) = 2096 \quad (4.4)$$

The amount of time t required for the processor to complete this number of cycles at the given operating frequency f can be calculated as:

$$\frac{\text{cycles}}{f} = t \quad (4.5)$$

$$\frac{2096}{1.92 * 10^6} = 1.0917 * 10^{-3} = 1091.7\mu\text{s} \quad (4.6)$$

Unfortunately, this is ≈ 3 times as long as the 320 μ s that the ISO/IEC 18000 standard requires, making our design incompatible with that standard. A method that some implementations use to work around this limitation is to queue up multiple tags at the same time and stage them to minimize the “dead” space on the RF medium.

Alternatively, the clock speed could be increased to the minimum speed necessary to make such an operation complete within the time requirement. To calculate the minimum clock speed, we use Equation 4.5 as before:

$$\frac{\text{cycles}}{f} = t \quad (4.7)$$

$$f = \frac{\text{cycles}}{t} \quad (4.8)$$

$$f = \frac{2096}{320\mu\text{s}} = 6.55\text{MHz} \quad (4.9)$$

In order to satisfy the ISO/IEC 18000 standard, the tag would need to operate

at 6.55MHz to perform an AES hash with sufficient rapidity. Even though this is relevant for dual purpose applications, our focus is on the EPC standard, which has a different timing restriction.

4.6.2 EPC Gen2

The EPC Class 1 Generation 2 UHF tag standard does not specify a single timing limit for the interchange between a reader and tag. The timing constraints are divided according to parameters that include the link frequency and frequency tolerance. These parameters are listed in Table 4.4.

The two major timing constraints related to the tag's interaction with the reader are T_1 , the time from reader transmission to tag response, and T_2 , the time from tag transmission to reader response. T_2 is also the reset timer for **Arbitrate**, **Reply**, and **Acknowledged** states shown in Chapter 2 Figure 2-1. T_{pri} is defined as the period of one cycle of the tag's Link Frequency (LF)¹⁴, which can vary between 40 and 640kHz.

$$T_2 = 20T_{pri} = 20 \frac{20}{\text{LF}} \quad (4.10)$$

This range of frequencies gives a maximum of between 31.25 and 500 μ s, comparable to the ISO/IEC standard. Since this constraint relates mainly to the reader's response, it is considered less of an issue.

We need to concern ourselves more with T_1 , which defines the maximum time a tag has to respond to a reader's request. Since any AES based commands will need to complete within this time period, we must ensure that the operating frequency is sufficient. Gen2 specifies a valid LF range of 40 – 640kHz. The frequency threshold (FT) varies between 4% and 22% on this LF range. We will assume that RT_{cal} ¹⁵ can be easily controlled such that it is not the limiting factor. Using Table 4.4, we can determine the time t allocated for a single TagCheck operation:

$$t = \max(RT_{cal}, 10T_{pri}) \times (1 + \text{FT}) + 2\mu\text{s} \quad (4.11)$$

$$t = 10T_{pri} \times (1 + \text{FT}) + 2\mu\text{s} \quad (4.12)$$

$$t_{hi} = \frac{10}{40\text{kHz}} \times (1 + 0.04) + 2\mu\text{s} = 262\mu\text{s} \quad (4.13)$$

$$t_{low} = \frac{10}{640\text{kHz}} \times (1 + 0.22) + 2\mu\text{s} = 21.1\mu\text{s} \quad (4.14)$$

A single request sequence must complete in 21.1 – 262 μ s. Using Equation 4.2, we discover this is a meager 40 – 503 operations per sequence.

Applying the new times to the result of Equation 4.8 for a hash-based TagCheck

¹⁴back-scatter modulation frequency

¹⁵the calibration time pulse from a reader to a tag

	Minimum	Maximum	Description
T_1	$\max(RT_{cal}, 10T_{pri}) \times (1 - FT) - 2\mu s$	$\max(RT_{cal}, 10T_{pri}) \times (1 + FT) + 2\mu s$	time from reader transmission to tag response
T_2	$3.0T_{pri}$	$20.0T_{pri}$	time required if tag is to demodulate interrogator signal, effectively time from tag transmission to reader response
T_3	$0.0T_{pri}$		time a reader waits after T_1 to send another command
T_4	$2.0RT_{cal}$		minimum time between reader commands

Table 4.4: Link timing parameters [EPC05a]

shows that the tag must run at 8.00 – 99.3MHz.

$$f_{hashhi} = \frac{2096}{f_{hi}} = \frac{2096}{262\mu s} = 8.00\text{MHz} \quad (4.15)$$

$$f_{hashlow} = \frac{2096}{f_{low}} = \frac{2096}{21.1\mu s} = 99.3\text{MHz} \quad (4.16)$$

Using the same approach to find the speed a symmetric TagCheck must run at yields numbers within a more reasonable range (3.94 – 48.9MHz):

$$f_{symlow} = \frac{1032}{f_{hi}} = \frac{1032}{262\mu s} = 3.94\text{MHz} \quad (4.17)$$

$$f_{symhi} = \frac{1032}{f_{low}} = \frac{1032}{21.1\mu s} = 48.9\text{MHz} \quad (4.18)$$

This indicates that the symmetric TagCheck is a better choice since it does not require as high an operating frequency. It is still not an optimal solution; higher operating frequency results in higher power utilization. We need a way to increase the time to complete an operation while still meeting timing requirements.

4.6.3 Command Interleaving

A simple mechanism to allow processing to occur in tight timing constraints is inspired from asynchronous communication protocols. In such protocols, a receiver is given a BUSY symbol by the sender while still processing data. We propose a similar idea; TagCheck tags that are meant to be used under the ISO/IEC or EPC Gen2 timing constraints will implement a **Busy** state. Any command that causes a tag to

enter **Busy** will be given a **HANDLE**¹⁶ that is used to discriminate between ongoing operations¹⁷.

Tags that are contacted by a reader using the **HANDLE** will immediately reply **BUSY**. This reply should be done by asymmetric transmission logic to minimize interruption of the main process and meet the timing requirements. Once the tag is done with its long operation, it will reply with the result to a reader that has the proper **HANDLE**.

This change to a tag's operation means that it can process long operations while still meeting the strict requirements of the ISO/IEC and EPC Gen2 specification. The only complication is ensuring that the **BUSY** logic is able to meet those strict requirements.

4.7 EPC Gen2 with TagCheck

The EPC Gen2 architecture was designed with the capability for expansion and proprietary augmentation. In this section, I will outline how to integrate TagCheck into the EPC Gen2 protocol.

Table 4.5 contains the EPC Gen2 commands, including the spaces reserved for expansion and the commands selected for TagCheck. To protect against garbling of commands, many are given a check-sum using Cyclic Redundancy Checks (CRC).

The EPC Gen2 specification divides tag memory into 4 different banks each with bit addresses 0x00 to 0xFF (Table 4.6). To use TagCheck, we will also need to make use of the reserved memory space allocated in Bank 00. Addresses 0x00 – 0x3F are taken by the passwords. The TagCheck secret needs 128 bits and we would like to minimize the requirement for extra memory. Therefore, 64 bits of additional secret are allocated at 0x40 – 0x7F¹⁸ with both addresses 0x00–0x7F treated as a combined secret¹⁹. For the hash-based implementation, additional 128 bits needed for the initialization vector are allocated at Bank 10 (the TID memory), from 0x7F–0xFF.

A TagCheck-enabled tag will add *Tagcheckh* and *Tagchecks* to the list of valid access commands for the **Acknowledged**, **Open**, and **Secured** state. A tag that receives *Tagcheckh* for the hash-based TagCheck will do the following:

1. Switch to bank 11.
2. Save the random 128-bit challenge *c* at 0x00–0x7F.

¹⁶Gen2 already uses handles for arbitrating commands.

¹⁷Current tags are single-threaded, but in the future tags may be able to handle concurrent operations.

¹⁸For an application that requires the secrets to be separated, the memory can be allocated at 40-AF.

¹⁹For a discussion of why 32-bit secrets are insufficient, refer to Section 3.6.1.

3. Switch to bank 10.
4. Load 0x7F–0xFF into the IV input of the hash function.
5. Switch to bank 00.
6. Load 0x00–0x7F into the data input of the hash function.
7. Switch to bank 11.
8. Load 0x00–0x7F into the data input of the hash function.
9. Transmit the hash value to the reader.
10. Return to the previous state.

A tag that receives *Tagchecks* for the symmetric encryption TagCheck will in turn respond thus:

1. Switch to bank 11.
2. Save the random 128-bit challenge c at 0x00–0x4F.
3. Switch to bank 00.
4. Load 0x00–0x7F into the key input of the encryption function.
5. Switch to bank 11.
6. Load 0x00–0x7F into the data input of the encryption function.
7. Transmit the encrypted value to the reader.
8. Return to the previous state.

This discussion completes the integration of TagCheck into the EPC Gen2 protocol.

4.8 Summary

TagCheck provides a simple and secure mechanism to prove the authenticity of a tag to a reader without sharing its secret with the reader. TagCheck requires a tag to have access to a block encryption function or secure hash function. We have synthesized a hash function using Feldhofer’s tiny-AES implementation [FWR05] according to Preenel *et al.*’s recommendations [PGV93]. Making use of the tiny-AES implementation requires an estimated minimum operating frequency of 3.94MHz.

By using a decoupled challenge-response Fiestel-like authentication, a reader is able to complete batch offline verification of multiple tags without access to the secret made use of by the tag and the verification service. Safe-keeping of the secret in this way is the key to minimizing the possibility of attack by a malicious or compromised reader. Preventing the reader from access to the secret eliminates leakage from the tag to reader and reader to service interfaces.

With the challenge of authenticating tags solved, a problem that still arises is that of tag modification and transference. Traditionally, such operations are enabled by transferring the previous secrets to the new owner. This action inadvertently exposes undesirable information to both parties. In the next chapter, I address this problem of secret leakage during ownership transfer of an RFID tag with the TagFolio system.

Command	Code	Length (bits)	Required?	Protection
<i>QueryRep</i>	00	4	Yes	Unique command length
<i>ACK</i>	01	18	Yes	Unique command length
<i>Query</i>	1000	22	Yes	Unique command length & CRC-5
<i>QueryAdjust</i>	1001	9	Yes	Unique command length
<i>Select</i>	1010	>44	Yes	CRC-16
Reserved	1011	-	-	-
<i>NAK</i>	1100000	8	Yes	Unique command length
<i>Req_RN</i>	1100001	40	Yes	CRC-16
<i>Read</i>	1100010	>57	Yes	CRC-16
<i>Write</i>	1100011	>58	Yes	CRC-16
<i>Kill</i>	1100100	59	Yes	CRC-16
<i>Lock</i>	1100101	60	Yes	CRC-16
<i>Access</i>	1100110	56	Yes	CRC-16
<i>BlockWrite</i>	1100111	>57	Yes	CRC-16
<i>BlockErase</i>	1101000	>57	Yes	CRC-16
Reserved for future use	1101001 1101111	40		-
Reserved for custom commands	11100000 00000000 11100000 11111111	-		Manufacturer specified
<i>TagChecks</i>	11100000 00011101	144	No	Not needed
<i>TagCheckh</i>	11100000 00011110	144	No	Not needed
<i>Busy</i>	11100000 00011111	24	No	CRC-16
Reserved for proprietary commands	11100001 00000000 11100001 11111111	-		Manufacturer specified
Reserved for future use	11100010 00000000 11101111 11111111	-		-

Table 4.5: EPC Gen2 commands with TagCheck [EPC05a]

Bank	ID	Description
11	User	User defined memory
10	TID	ISO/IEC 15963 allocation class identifier and additional data
01	EPC	EPC data, PC CRC-16
00	Reserved	Access and Kill passwords

Table 4.6: EPC Gen2 memory map [EPC05a]

Chapter 5

Tag Modification and Transference

“If you wake up at a different time in a different place, could you wake up as a different person?” [Pal96]

This chapter describes TagFolio, a mechanism for applying a person’s privacy policy to the RFID tags they possess and simplifying the method for transferring tag ownership. Any change in security policy is applied concurrently at the time of the transfer. TagFolio allows transfer of ownership with minimal change to the RFID tags and no leakage of long-term secrets. It also allows a tag to continue operating in a secure mode when TagFolio is not present, unlike the RAT [SB06] or Watchdog Tag [FSL04].

Secret initialization and transfer is a problem not often addressed in discussions about cryptographic systems. Some solutions assume that an out-of-band mechanism is used for key synchronization such as manually copying down a number written on packaging or using a hardwired connection. These out-of-band mechanisms are often questionable or inelegant.

Even the application of public-key cryptography does not solve the key initialization problem. Our two users who wish to communicate, Alice and Bob, must have some level of assurance in the public key they acquired during their initial exchange in order to have any security. A man-in-the-middle attack can be used to monitor the data that passes between two such parties if there is no additional mechanism to associate keys with identities.

Traditionally, transfer of ownership is accomplished by transferring the previous secrets to the new owner. This action leaks a significant amount of information to both parties. In the case of the new owner, the previous passwords may be applicable to other tags or be used to derive the passwords for other tags. Optimally, each tag has a different password, but it can be expected that this will not be universally enforced or even realistic in some environments. In the opposite direction, simply giving secrets to the new owner does not prevent the old owner from continuing to utilize the associated access until the new owner changes such secrets if applicable. There is no mechanism for forcing the previous owner to forget such information. To

minimize the window of opportunity, the secrets would be changed at the time of the transaction, but how can that be done without exposing them to the previous owner. All of these issues are addressed in TagFolio .

TagFolio only requires an increase in tag storage and a small cryptographic cipher as opposed to the addition of strong cryptographic protocols. TagFolio specifies the design of a separate device that takes care of the cryptographic operations and interfaces securely with tags by using small additional storage areas. This chapter builds upon the hash function synthesis discussed in Chapter 4.4.3 with the assumption of symmetric block cipher and cryptographic hash function availability.

Unlike much of the rest of my thesis, TagFolio is designed to be used independently of the security mechanisms of the EPC Gen2 standard. This is because incorporating TagFolio into the Gen2 standard would require a significant rewrite of the entire system. Instead, TagFolio is designed as a replacement for much of the **Open**, **Secured**, and **Acknowledged** states.

TagFolio addresses the initialization and transfer of secrets by employing an authorized intermediary. The intermediary uses its authority to complete transactions on a tag as well as apply the new owner's blinded authorizations to the tag.

5.1 TagFolio

As mentioned previously, TagFolio addresses the problem of application and verification of an owner's privacy preferences to the RFID tags they possess. Possession implies that the owner should have access to all relevant data and keys associated with their tags, though an in-depth discussion of the definition of possession is included in Section 6. In addition, TagFolio also addresses difficulties surrounding the transfer of the tag's ownership.

A standard EPC Gen2 tag allocates 96 bits for its ID, 64 bits for its two passwords and some small amount of working registers. The tag I propose has 2 96-bit ID storage areas and 3 Hash lock 128-bit storage areas, for a total of 576 bits of storage. This is well within the 1,024 bits of addressing space specified in the Gen2 standard [EPC05a].

TagFolio uses the JanusTag mechanism described in Section 4.1. The hash lock areas must be Read-Write, same as the PID.

5.1.1 TagFolio Structures and Requirements

The TagFolio tag is capable of 6 operations outlined below ($L()$ represents a hash lock):

1. Receive $L_1(NewID)$: Change PID to the following argument
2. Receive $L_2()$: Return RID
3. Receive $L_3(CHANGE, l_1, l_2, l_3)$: Change hash locks to arguments 1,2,3

4. Receive $L_3(\text{SILENCE})$: Silence tag¹
5. Receive $L_3(\text{KILL})$: Kill tag
6. Normal Query: Return PID

The three hash locks are arranged according to progressively more restrictive access privileges, intended to provide rough granularity of access control. To emulate EPC Gen2 behavior, only 2 hash locks would be needed: one for unlocking the tag and one for killing the tag. For simplicity, the algorithm could be compressed to require only one hash lock to control all access, but the lack of granularity would limit our application. A TagFolio-enabled tag's 3 hash locks divide operations into the most common types of access: changing its disguise, revealing its true identity, and giving it a new commander/ ending its life. In order to interface with a TagFolio-enabled tag, a TagFolio instantiation needs:

- Secure storage of the user's secret S_u , only accessible via certificates
- Have encryption capabilities: $E_{\text{key}}(\text{msg})$
- Have cryptographic hash-function capabilities: $H(\text{mesg})$
- Optional: publicly Write-Only memory for logs²
- Accessible credit card entries with optional PIN
- Interface for a secure connection (USB, Firewire, LAN, 802.11b)
- Cryptographically secure pseudo-random number generator

Given the capabilities of the TagFolio and it's associated tag let us consider their interaction in the form of protocols. Most applications relating to ownership transfer divide into 3 environments discussed one-by-one in the following sections. The environments are:

Exchange: visibility most important

Transport: security most important

Home: needs to be user defined

¹Silencing makes a tag temporarily unresponsive until specifically addressed and sent a valid hash lock.

²Such memory would also be used for electronic receipts as described in Section 5.1.5.2.

5.1.2 Exchange

In an environment where tags are being exchanged (such as a commercial or industrial setting), visibility is important. However, there is also a concern of foreign agents gathering too much information. This necessitates having a separate ID (called STOREID) used internally for partitioning the information as the security policy dictates.

The STOREID is an internal ID that the store-room uses for its internal inventory and transaction processes, such as Point of Sale (POS). This ID should use ONS to point to relevant information given the item's current environment. This will be described in detail in Section 5.1.2.3. The static and private RID is used to map into the store's database and extract ($\text{StoreDB}_{\text{get}}()$) the key used to lock the tag.

5.1.2.1 Tag Setup

When a tag enters a store through the loading dock, it contains a set of hash locks pre-programmed by the manufacturer and given to the store. The store uses a protocol detailed in the next section to change the hash locks to values in the store's database keyed off the tag's RID. The $H()$ function used in this process can be any of a number of cryptographically strong hashes. I suggest the AES hash described in Section 4.4.3 as an example. At the end of the store initialization, the tag's state should be as shown in Table 5.1.

PID	STOREID
RID	RID
$L_1()$	$H(\text{StoreDB}_{\text{get}}(\text{RID} + 1))$
$L_2()$	$H(\text{StoreDB}_{\text{get}}(\text{RID} + 2))$
$L_3()$	$H(\text{StoreDB}_{\text{get}}(\text{RID} + 3))$

Table 5.1: TagFolio tag initialization by the store

5.1.2.2 Operation

After initialization, the tagged item is placed inside the store. The next major step in the item's lifecycle is being purchased or transferred out of the facility. The new prospective owner places the item on their cart/forklift, takes it to the cashier/transfer point and connects TagFolio to the register/reader. As we describe the operations that follow as the item is passed over the reader/writer, refer to the conceptual drawing of TagFolio in Figure 5-1 and the pseudo-code representation of the tag's main control function in Figure 5-2:

1. The reader $\mathcal{R}$ queries a tag $\mathcal{T}$.

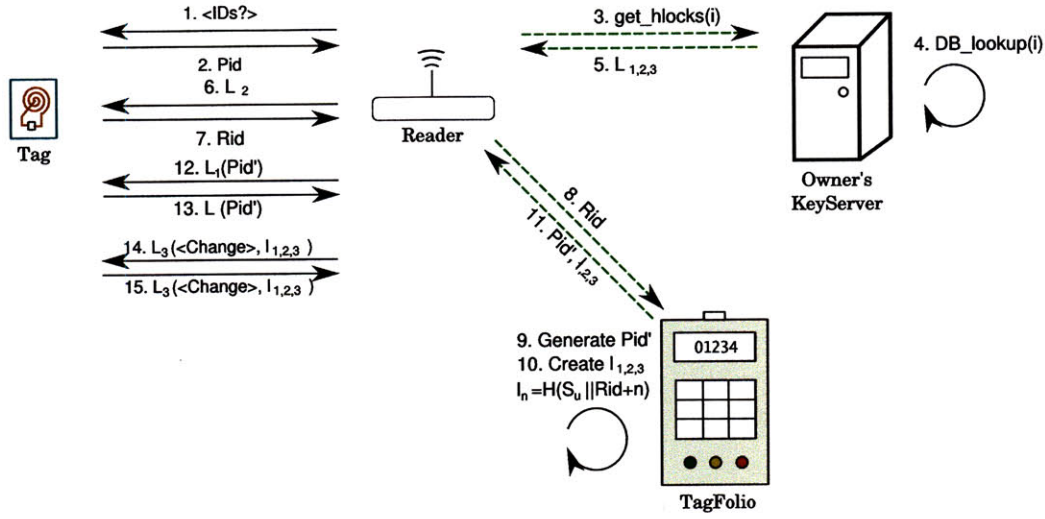


Figure 5-1: Concept drawing of TagFolio

2. The tag $\mathcal{T}$ returns its PID.
3. $\mathcal{R}$ sends the PID to the owner's key server $\mathcal{S}$ with a request `get_hlocks(i)`.
4. $\mathcal{S}$ looks up hash locks in a database using the PID as a key.
5. $\mathcal{S}$ returns $L_{1,2,3}()$ to $\mathcal{R}$.
6. $\mathcal{R}$ sends $L_2()$ to $\mathcal{T}$.
7. $\mathcal{T}$ verifies $L_2()$ and returns its RID to $\mathcal{R}$.
8. $\mathcal{R}$ sends RID to the TagFolio $\mathcal{F}$.
9. $\mathcal{F}$ applies its recoding to RID, generating PID'.
10. $\mathcal{F}$ generates the new hash locks $l_n = H(S_u || RID + n)$ where $n = 1..3$.
11. $\mathcal{F}$ instructs $\mathcal{R}$ to program the new PID' and to install the new hash locks l_n .
12. $\mathcal{R}$ programs PID' by sending $L_1(\text{PID}')$ to $\mathcal{T}$.
13. $\mathcal{T}$ responds with $L_1(\text{PID}')$, indicating success.
14. $\mathcal{R}$ programs the hash locks by sending $L_3(\text{CHANGE}, l_1, l_2, l_3)$ to $\mathcal{T}$.
15. $\mathcal{T}$ responds with $L_3(\text{CHANGE}, l_1, l_2, l_3)$, indicating success.

At step 10, the pre-configured preference of the user is applied to PID by choosing to execute one of the following:

```

TAGFOLIOTAGDISPATCH( $L, M, l_1, l_2, l_3$ )
1  if  $L == \text{NULL}$ 
2      then return  $Pid$ 
3      else                                     ▷ Check for hash locks
4          if  $L == L_1$                          ▷ Change public identity
5               $Pid = B$ 
6          return OK
7  if  $L == L_2$                                  ▷ Show real identity
8      then
9          return  $Pid$ 
10 if  $L == L_3$                                 ▷ Full access
11     then
12         if  $M == \text{CHANGE}$ 
13             then
14                 for  $j \leftarrow 1$  to 3
15                     do  $L_j = H(l_j)$ 
16         if  $M == \text{SILENCE}$ 
17             then
18                  $Mode = \text{SILENCED}$ 
19                 return OK
20         if  $M == \text{KILL}$ 
21             then
22                  $Mode = \text{KILLED}$ 
23                 return KILLED

```

Figure 5-2: Pseudo-code representation of a TagFolio tag operation

- Kill: return KILL code
- Silence: return SILENCE code
- Zero: return zeros
- Randomize: return a random number
- Encrypt: $E_{S_u}(\text{RID})$
- User Defined: starting at 0, return an increment number
- Expose: return RID

Assuming the procedure completed successfully, when the user goes to the gateway leading to the outside world or another area, the gateway reader will now see one of the following:

- no tag response (KILL or SILENCE)
- a tag with the serial number 000...
- a random number
- $E_{S_u}(\text{RID})$
- a constantly increasing serial number
- RID
- a custom number like “31337B00B5...”

For secondary verification that the tag was actually rewritten, we can ask the gateway reader to verify the tags against what the reader at the register set them to be. This eliminates false negatives and gives better feedback to all involved. Such feature can even be a selling point indicating that the service is to help ensure the user’s privacy preferences are being fulfilled. A TagFolio might have its own reader for a final verification.

For added customization, a small keypad and numeric display can be installed on the TagFolio to allow a PIN entry requirement before use. Other biometric authentication mechanisms could also be used. The keypad would allow the user to pre-program security settings for numerous situations. For example, while the user may leave grocery lists exposed, they may wish to conceal the medication they buy at the pharmacy.

As a design strategy, the hash locks are only ever sent over the air once before they are changed. Since they are one-time-use passwords, this is a necessary requirement. Any additional use other than to change the hash locks will lead to significant reduction in security.

The TagFolio has significantly more computational and electrical power allowing the new PID can be arbitrary. The only timing constraint that must be satisfied is having the challenge and response complete while the tag is still in the reader’s field. Very fast and small chips for doing both public [DM02] and private cryptography exist [FWR05] that are suitable for such applications.

Each of the above customizations might be of interest to a different focus group as mentioned by [dJ06]. The Privacy Fundamentalist, who is most concerned with privacy no matter what the cost, will select the Kill option every time. Unfortunately, by doing so, such user will eliminate any benefit from home automation or recycling systems (recall Bob’s story in Section 3.3.3.1). On the other side of the spectrum are those who are completely unconcerned about privacy or who don’t have a TagFolio.

These users might have their tags programmed with the RID, which allows applications to work, but provides no privacy protection. I will assume that the majority of our users are Privacy Pragmatists, who choose encryption to protect privacy while still gaining post-gateway benefits. Such individuals are the ones who most need to be convinced to make use of such technology. Other users will choose convenience or simplicity over privacy making their choice more clear.

A layer of security that allows TagFolio to be secure enough for both audiences is to remap the IDs. In the next section, I discuss the details of the remapping process that divides internal and external IDs.

5.1.2.3 Remapping

While the item is in the store, accesses the STOREID for the product which is publicly broadcast as PID. The store would need to have an entry in ONS that simply forwards any query to the ONS server that contains information about that product.

For instance if a tag's RID was composed symbolically as:

```
VERSION.PROCTERANDGAMBLE.PAMPERS.00000001
```

which pointed to Procter & Gamble (PG.COM), the store (let's say Walmart) might turn the tag's RID into:

```
VERSION.WALMART.PNGSHORTINTERNAL.PAMPERSINTERNAL.INTERNALSTOCKNUM
```

which, if queried in ONS, would take you to `mirror1.walmartepcisproxy.com`.

Doing an EPC-IS query such as:

```
http://mirror1.walmartepcisproxy.com/lookup?id=
VERSION.WALMART.PNGSHORTINTERNAL.PAMPERSINTERNAL.INTERNALSTOCKNUM
```

would cause ONS to look up the RID from the given ID, and do the following query:

```
http://epc-is.pg.com/lookup?id=
VERSION.PROCTERANDGAMBLE.PAMPERS.00000001
```

which will actually return the relevant information for that product.

An alternative scheme is to simply remap only the first two domains:

```
VERSION.PROCTERANDGAMBLE.PAMPERS.00000001
```

In this scenario, the Walmart server would need to somehow relay the information that PNGSHORTINTERNAL is actually PROCTERANDGAMBLE.

A third scheme is to make use of the PTR³ record in DNS/ONS. In this case, Walmart's ONS server would simply return a PTR record to a resolver, which would give the correct name for a secondary resolution. To illustrate this, resolving:

³Short for pointer, a record that points at another.

VERSION.WALMART.PNGSHORTINTERNAL.PAMPERS.00000001

would return the following PTR record:

VERSION.PROCTERANDGAMBLE.PAMPERS.00000001

The resolver would then do a secondary resolution in order to gain access to the EPC-IS server containing information about the product.

Interestingly, many large retailers already require vendors to remap the name-space of their bar codes to be compatible with their particular flavor of inventory system and POS. This indicates that ID remapping, though more technical, is a familiar concept. Therefore, converting vendors may be easier than users in other domains.

Regardless of what the retailer does, as long as the number is unique for the given storage area/store, the final remapping will not work when the item is removed from the said area. A potentially confusing situation may arise if the same remap IDs exist in multiple domains. However, this can be solved by setting aside numbering spaces, much like the 192.168.*.* address is reserved for local home networks. These IP addresses do not resolve to real IP addresses outside of the local network.

5.1.3 Transport

During transport, a TagFolio encrypted tag will be programmed with the values shown in Table 5.2.

PID	$E_{S_u}(\text{RID})$
RID	RID
$L_1()$	$H(S_u \text{RID} + 1)$
$L_2()$	$H(S_u \text{RID} + 2)$
$L_3()$	$H(S_u \text{RID} + 3)$

Table 5.2: TagFolio tag state during transport

As mentioned previously, without knowing the hash locks, the only information accessible to a reader is $E_{S_u}(\text{RID})$. Furthermore, without knowing S_u , recovering the RID is impossible.

5.1.4 At Home

Once the new owner of a tagged item arrives at home, the system is designed to allow the the tag's ID to be decrypted for use with home automation appliances (*e.g.* a refrigerator). In addition, silenced tags can be instructed to resume transmitting.

To decrypt the tags, the user would connect TagFolio to their automation controller. This computer would download all of the RIDS stored in the TagFolio. Alternatively, if the computer knows S_u , it can simply decrypt a given tag by trying $D_{S_u}(\text{PID})$ and checking for a sensible result. This is another validation check to make sure that the privacy policy of the TagFolio was applied correctly.

Returning to the realm of pharmaceuticals, let us consider a medication that needs to be taken every 4 hours with serious consequences if not taken on schedule [KFR06]. Given a small amount of automation and ubiquitous readers, the house can remind the patient to take the medication at appropriate times. The controller can also be used to ensure that there are no contraindications, and to locate the medication if the patient misplaces it.

5.1.5 Further Applications and Augmentation

Given its architecture, it is possible to extend TagFolio's capabilities to other types of ownership transfer including the traditional commercial purchase.

5.1.5.1 Contact-less Credit Cards

TagFolio can be coupled with other systems. We will illustrate this with an example of contact-less credit cards. American Express began putting RFID contact-less Smartcards on their latest credit cards for use with the ExpressPay system [Ame06]. Interestingly, unlike most RFID contact-less Smartcards, American Express chose to make the circuitry and antenna explicitly visible as a marketing and design feature.

The ExpressPay system is designed to make purchasing convenient, but it does reduce security because it eliminates the requirement for a signature. Consequently, concerns of abuse will limit initial usage to low-priced ephemeral purchases such as fast food.

Taking a lesson from [RSA06], TagFolio could act as a secure storage for credit card information to make the RFID more intuitive by association. If the functionality of the American Express "Blue" contact-less Smartcard was integrated into a TagFolio, undetected theft would be much more difficult.

To understand how TagFolio would improve security in this situation, let's assume that TagFolio is normally left locked, such that it does not respond. Once the owner unlocks TagFolio, the Smartcard-module is ready to respond either wirelessly, via USB, or the traditional Smartcard contacts. When the user is done making a payment or when an internal time-out is reached, the system re-locks itself. This is similar to the locking feature many PDA's and cellular phones use.

If a thief acquired the card while it was locked, the card would be useless to him. If he acquired the device while unlocked, it would only be useful until the PIN was requested again. In terms of configuration, this allows a concerned user to specify a very short window of decreased security, while allowing an unconcerned user who

prefers convenience to set the window to much larger. Additionally, a very concerned user could set the TagFolio to never use the contact-less payment method. The most concerned users might worry about secret extraction via side-channel attacks as described in Section 5.2.2, but this is unlikely to be a problem for most applications.

5.1.5.2 eReceipts

One academic project I worked on⁴ focused on streamlining the store checkout process. Elimination of paper receipts became the core of the project. After examining many different methods of compressing, scanning, and rolling receipts, one of our team members came up with the idea of eliminating the paper element altogether. Thus the idea of an eReceipt was born.

Our final demonstration was a wireless key fob with 2 buttons. The device communicated via IR with a reader connected to a computer acting as a cash register. When the “register” completed its transactions, the user pressed one of the buttons on the fob. The fob initiated a handshake and the computer uploaded the receipt information in text format to the fob. The information could be downloaded later to the user’s home computer for storage or printing.

TagFolio can function in much the same way as the key fob, simply replacing the IR interface for the RF link. This would require adding additional protective storage space to the TagFolio. The storage area would be write-only in the locked mode. When the device was unlocked, the storage area could be read. This concept is similar (though inverted) to the read-only switches added on to some USB storage drives. An additional port could also be placed on the TagFolio to enforce a security policy on flash media data for receipts.

5.1.5.3 Emergency Information

Portable storage devices have already greatly changed peoples’ lives. Nowadays people routinely exchange electronic business cards via IR on their phones and PDA’s. With the ubiquitous nature of wireless devices and their large data storage capacity, the logical next step is storing important emergency information on them. Such information is useful directly to the user when, for example, their wallet is stolen and they need to cancel their credit cards. However, emergency information becomes pivotal when the user is unconscious or in an altered mental state.

According to the Massachusetts Basic Emergency Medical Technician protocol [Sta05], the basic emergency information needed breaks into 6 categories with the acronym SAMPLE:

Signs: what the EMT observes, not likely to be predetermined

Allergies: particularly to medications like aspirin or penicillin

⁴as part of 2.739J Product Design and Development

Medications: to determine contraindications for additional medications and procedures

Pertinent medical history: to help with assessment

Last oral intake: to determine the length of time since the person last ate/drank something, particularly if surgery is required

Events leading up to the incident: to examine environmental effects and any other telltale signs

In addition, all standard EMT paperwork needs to include information such as:

- full name
- address
- phone number
- primary care doctor
- emergency contact - name and phone number

A program deployed at several assisted care facilities included putting the above emergency information in a standardized place, such as inside kitchen cabinets. One such program called “Vial of Life” used a standardized medicine bottle containing all the information to be kept in the refrigerator.

TagFolio could be used to achieve the same effect. Similarly to the military dog-tags discussed in Section 3.3.1.1, a patient could store important contact information and an encrypted set of medical records on their TagFolio. This information could be unlocked using the same mechanism previously described. EMTs could be given access to a key for decrypting relevant medical information.

A casual reader might ask why EMTs do not simply access a plastic card placed inside a person’s physical wallet. The reason is conjoined with the very motivation behind TagFolio: personal privacy. Going through a patient’s pockets looking for a wallet is an invasion of privacy and might be viewed unfavorably if anything from the pockets or the wallet itself went missing. In addition, some patients have dangerous items such as used syringes in their pockets. Considering these factors, a standardized solution such as MedicAlert⁵ [Med] is far superior.

5.2 Analysis of TagFolio

Now that we understand the design and operational environments of TagFolio, let us analyze the system’s strengths and weaknesses.

⁵For a small fee, MedicAlert provides a necklace that displays important medical information and a unique ID number that can be used to retrieve further medical information.

5.2.1 Cost Analysis

As always, we wish to minimize the tag's additional cost and requirements for architectural redesign. Most RFID security systems require a significant amount of layout space to accommodate the necessary cryptography, which can add significant cost.

TagFolio removes the responsibility of security management from the reader by instructing it how to reprogram the tag. A user only needs to trust that the reader can read and write tags according to TagFolio's instructions. Such simple readers are cheaper to implement and maintain.

To prevent secret leakage distrust is mutual during an ownership transfer; TagFolio is not presented with the secrets previously used to access the tag and the previous owner is not shown the new secrets used to lock the tag. This allows hash locks to be reused in controlled circumstances. In the case where reuse of secrets is desired, such a system can reduce cost by reducing the infrastructure for storing the secrets. In addition, the cost of an inadvertent exposure of secrets must be taken into account to compare with TagFolio's hash locks which are designed to hide the exchange from the unauthorized party.

As mentioned in the design discussion, the user has an opportunity to audit the system. Cash registers already display the UPC code when items are scanned. On RFID systems, displaying the EPC code both before and after TagFolio's processing would help reassure the user of TagFolio's proper functioning and reduce errors which impact efficiency.

Lastly, TagFolio eliminates the need for separate kill kiosks. The readers at the cash register can complete kill requests, thus reducing the additional cost associated with having such kiosks.

5.2.2 Threat Analysis

As described in [KJJ99], a worry with any Smartcard-style implementation is power analysis. Using a technique called differential power analysis, Kocher *et al.* were able to recover a variety of secret information from devices including keys. This information was visible by the changes of power-consumption on the device because each instruction used a different configuration of transistors. Needless to say, such techniques might be employed on the TagFolio, so care must be taken to make such analysis more difficult. Kocher *et al.* recommends:

“...reduce signal sizes, such as by using constant execution path code, choosing operations that leak less information in their power consumption, balancing Hamming Weights and state transitions, and by physically shielding the device.”

Many of these solutions require significant changes to the chip's design, but some need only slight modifications to the chip layout software. A combination of these and newer techniques could be used to make TagFolio safe from power analysis attacks.

If TagFolio were to carry financial information equivalent to a credit card, other avenues of attack could open up. This is described by Ferguson *et al.* in [FS03a] when talking about the SET on-line credit card system designed to protect on-line shopping with a credit card. SET encrypts the credit card information so that an eavesdropper cannot gain access to it. Instead of a physical signature, SET uses a digital one generated by the SET enabled card. This should be more secure, but it changes the liability basis for such transactions. To understand why, let's examine what happens during a credit card fraud claim.

When credit card fraud occurs, the victim complains to the company and disputes the charges. The company contacts the stores that accepted the charges and requests copies of all signatures. Copies of the signatures are then sent to the victim to compare against their own. The victim gives a description of why they believe the receipts and signatures were not their and returns the paperwork to the credit card company. The company examines the claim and makes a ruling.

In my own experience, this system works. I had my card stolen in September 2001. When I discovered unknown \$1,000 charges of random items on my card, I was worried. I complained to the credit card company and was presented with copies of all the receipts. I discovered that the thieves were not very smart, but the cashiers were worse. The signatures that were in the copies did not resemble my signature on the back of the card at all. In some cases, my name was very clearly misspelled. Signatures as a form of non-repudiation, though not efficient, were effective in this case.

With a TagFolio or SET system, a physical signature does not exist. Users need not worry about someone forging their signature. However, they do need to worry about Trojan horses and viruses attacking the computer that contains their credit card information or equivalent secret, and using this privileged information to make irrefutable purchases (irrefutable because the digital signatures are guaranteed to match).

To combat such exploits, privileged information should be secured with encryption on the computer it is stored on. This may not be enough, however, since social engineering methods such as phishing will extract privileged information from information even when properly encrypted.

Another weakness of TagFolio is its requirement for a compatible reader at the Point of Sale (POS). Unless the TagFolio concept becomes a standard that designers of RFID-enabled POS care about, market penetration will be difficult. Addressing this problem will require good marketing and proof of necessity.

5.3 Summary

In this chapter, we explored the design of TagFolio, a system for securely transferring ownership of an RFID tag. Previous systems have addressed this problem by simply

transferring the secret contained on the tag. This simplistic model opens up a wide variety of technological and social attacks.

TagFolio's major innovation is using an intermediary to apply blinded secret updates to the tag. These blinded secrets are generated using Weis' hash locks [Wei03]. The one-way property of hash locks allow updates to occur without exposure of the previous secrets to the new owner, and without giving future access to the previous owner. The TagFolio mechanisms can also be used to interact with other financial transactions making use of static secrets. In the next chapter, we turn to the next phase of a tag's life – manipulation of its secrets and its demise.

Chapter 6

Tag Access Control

“They’ve never met so Tyler thinks it’s a bad thing that Marla is about to die.

It’s nothing of the kind.

This is none of Tyler’s business, but Tyler calls the police and Tyler races over to the Regent Hotel.

Now, according to the ancient Chinese custom we all learned from television, Tyler is responsible for Marla, forever, because Tyler saved Marla’s life.

If I had only wasted a couple of minutes and gone over to watch Marla die, then none of this would have happened.”[Pal96]

“On a long enough time-line, everyone’s survival rate drops to zero.”[Pal96]

The major challenge with all cryptographically enabled RFID security systems is managing the various secrets that the tags, servers, and readers need. The TagCheck protocol we discussed in Section 4.3 requires a secret on both the server and the tag. TagFolio from Section 5.1 is used to manage the various secrets including those needed to kill or reprogram the tag. In this chapter, we combine TagCheck and TagFolio to design an access control scheme called TagDirective that is fast, secure, robust, and easy to set up.

In particular, we need an access control mechanism capable of enforcing time limits on access while not exposing static secrets. In many authentication systems, access is assumed to last until the secret is changed, which eliminates the possibility of giving temporary rights to another individual. Such all-or-nothing schemes are extremely limiting and insufficient for our needs.

In addition to access control management, we need to address the problem of how tags should be killed. We need to protect large numbers of tags from being killed by malicious attackers that have discovered an exploit or gross mis-configuration. To do so, I propose a number of different kill protocols that make mass kills harder while still allowing authorized kill operations to occur. Consideration of the kill protocols provides further context for TagDirective’s operation.

6.1 The Mass Kill Problem

A scenario to be mindful of for any RFID technology is that of “mass killing”¹. I prefer the term “machine-gun killing” and will use the two interchangeably². To execute a “mass kill”, an attacker first finds an exploit that allows easy access to a non-physical kill mechanism for an entire class of tags. He then goes into an area where the tags have been deployed and begins rapidly killing them. The motivations behind such an act could lie in the area of mischief or malevolence. Regardless of the motivation, though, such attacks need to be prevented.

6.1.1 The Kill Protocol

To illustrate what happens when a kill command is issued, Figure 6-1 shows a simplified flowchart of an EPC Gen2 Kill sequence. Depending upon what a manufacturer determines is the proper “kill state” for a tag, the killed tag could end up in any of the following states:

Lock: don’t talk, but don’t change internal state

Erase and lock: don’t talk, but do erase internal registers

Erase and throw away password: erase internal registers, don’t talk, and set the password to an invalid setting

Randomize and lock: don’t talk and set internal registers to a random bit-string

Burn: self-destruct³

Reduce range: change impedance of the antenna or reduce transceiver sensitivity to force a stronger and hopefully physically nearer interaction

6.1.2 Kill Information Management

Assuming that a sufficient protocol for killing the tags is developed, we are still faced with information management issues. The major problem comes simply from storing and managing all of the kill passwords. Difficulties that fall into this category include incorrect entries in the password database (or on the tag) and managing access control to the database. If the tags cannot have their kill passwords programmed, then passing kill information to the next entity controlling the tags is also an issue.

¹The phrase “mass killing” comes from common usage in many of Sarma’s works [SF06].

²The phrase “serial killer”, as a pun on the tag’s unique serial number, would be flashier, but not as appropriate.

³which I like to call “hitting the big red button”

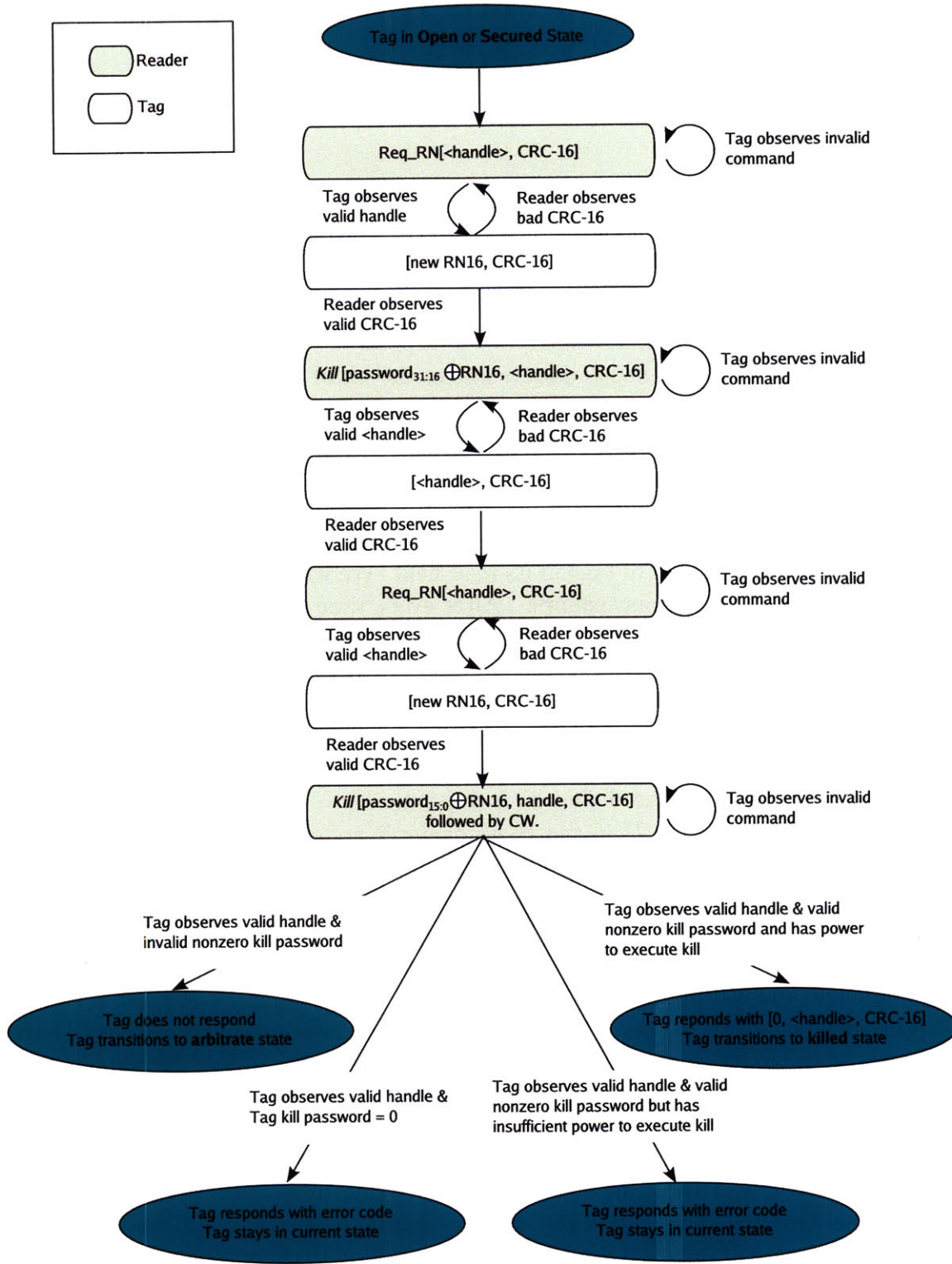


Figure 6-1: EPC Gen2 kill flowchart

6.2 Protection from Mass Killing

To protect tags from mass kill attacks, I designed a number of mechanisms, described in the following sections.

6.2.1 Reader Misdirection

Many attacks assume that a reader only talks to valid tags. We can take advantage of this assumption and instruct a reader to perform fake reads in order to confuse the attacker. This is easily accomplished as the reader has significantly more power, both transmitting and computational, at its disposal.

The idea of using extraneous data is known as “chaff” and was first described in [Riv98]. Assuming bandwidth has not been fully utilized, we can use chaff to lower the signal-to-noise ratio and consequently keep a malicious attacker from simply listening to gain the information they seek. Thus, chaff allows a reader to draw out an attacker by emulating tags that do not exist. We call this system Reader Misdirection.

Let us illustrate the idea of Reader Misdirection with an example. Assume that a reader $\mathcal{R}$ has some number of tags in its read field. We will focus on two of these tag, T_1 and T_2 , for the sake of our example. After every time period t , the reader queries its field to see what tags are present. Using the tree-walking anti-collision protocol, T_1 gets read, followed by T_2 . Departing now from the usual procedure, the reader follows the anti-collision tree for a virtual tag T'_3 , knowing that it will not receive a valid response. The reader remembers T'_3 and will continue to query it each time it checks all of the tags in its field. Any attacker who is listening to the high-power reader signal will be convinced that T'_3 is present.

Virtual entities such as T'_3 in the above example are extremely useful in a variety of situations. An attacker, who wishes to mass kill tags, has a chance of picking a tag that does not exist. When a friendly reader notices this condition, it can begin to engage any of the jamming procedures described later in Section 6.2.2 to protect the remainder of the real tags. These non-existent tags also make tracking harder for any malicious monitoring agent.

One challenge with Reader Misdirection is deciding how many virtual tags to have, and how often to change them. Optimally, this would be determined by watching the typical behavior of tags, then creating a model for the reader to follow such that an outside monitoring agent would not be able to tell the difference. As a starting point, I would recommend a 2:1 ratio of real to virtual tags with the virtual tags changing approximately 1/3 of the time to maintain the same proportion. Needless to say, if everyone used the same ratios and change timing, a listener could back-solve the correct proportion of tags and use Gaussian elimination to determine which tags were real. My recommendations are meant only to provide a guideline.

A significant advantage of Reader Misdirection rests in the requirement for no modification to any of the existing tag systems. Even the modification to the readers

themselves is minimal – a simple addition of a function that emulates the reading of a non-existent tag. In order to emulate a tag, the reader needs to understand the tag’s end of the protocol, but such modification is also trivial.

6.2.2 Listening for “Gunfire” (Neighborhood Watch)

We can also try to throttle mass kill attempts by using the jamming concepts developed in Section 2.1.8. In an environment with friendly readers reading the tags before and during the slaughter, mass kills could be detected similarly to theft prevention. To do so, we assume that a separate reader-to-reader mechanism is used to differentiate reader authorization. When the friendly readers notice a number of tags suddenly disappearing, they should realize something is wrong and engage countermeasures. What we’re doing here is essentially creating a community of friendly readers, which I will refer to as a “neighborhood” or “hood”, with the capability to listen for “machine-gun” readers give unauthorized kill commands and respond to it.

The neighborhood could make use of a PG-like web of trust for its key-exchange protocols and event messaging systems. Once initialization is complete, the hood can operate as a unit. When one reader notices a questionable event, it can notify other readers to take action. These actions could request human intervention or enable kill-resistance methods requiring human activation.

The challenge with this type of a system lies in discriminating between authorized and unauthorized reader operations. We could have the hood be given commands via a separate back-end network publish-subscribe protocol to indicate which new readers are acting in accordance with local policy. However, I proffer that such a channel is unnecessary, as this information is already being broadcast via the killer’s kill request. If desired, we could also append authorization information to the kill operation as a warning to the hood. I call this method of broadcast secondary authorization “Neighborhood Watch.”

For an example of Neighborhood Watch, let us consider an authorization token v_1 . If a kill authorization server $\mathcal{K}$ has a well-known public key p that has been distributed to the hood, it can sign the authorization and concatenate it to the end of the message in token v_2 :

$$v_2 = v_1 || S_p(v_1)$$

Unfortunately, this does not give any time context for the authorization or even what authorization has been given. We provide context by including the time t and the operation specified in our final token v_3 .

$$v_3 = v_1 || S_p(v_1 || t || \text{KILL}) || t || \text{KILL}$$

This new token prevents an attacker from changing the time or expected operation, while still allowing the tag to access the token. When authorized readers see the authorization request go by, they will know that the kill is authorized and should be

allowed. If a kill request without a valid signature is detected, the readers will take measures to increase the killing difficulty by making the time delay parameters of the kill resistance mechanisms more punitive. The specific kill resistance mechanisms I developed are described in the rest of this section.

6.2.2.1 Temporary Reprieve

In my first kill resistance concept known as “Temporary Reprieve”, killing a tag is a two-stage process. A flowchart of the Temporary Reprieve mechanism is shown in Figure 6-2. A tag that is supposed to die will be given a kill command, then left powered for some nominal amount of time such as 1 second (timer n). Once this time elapses without interruption, the tag becomes permanently silent. However, if during the nominal time the tag receives the command `REPRIEVE`, it does not kill itself.

The Temporary Reprieve concept solves the machine-gun killing problem with minimal required modifications. The only additional requirements are the ability to recognize the `REPRIEVE` command, and a reliable internal timer. We cannot use an external timer such as the modulation of the transmitter, because a malicious reader might be able to change the frequency to speed up the waiting period. Conveniently, most RFID tags already have an internal clock signal that can be used for this purpose.

Needless to say, Temporary Reprieve can be misused to prevent tags that should be killed from dying, but post-verification with a separate read transaction can discern if this is the case. An active attacker could also still accomplish his task of killing many tags by sending out a request to kill a tag, then jamming the channel until the tag finishes its dying process. While this makes the attacker’s job more difficult, it still creates a vulnerability we would like to patch. To make jamming a less viable option, we must first develop a method of ensuring that time constraints placed by the tag on execution are inviolate. This method, called a Deadline Timer, is introduced next.

6.2.2.2 Deadline Timer

Implementing an inviolate timer is not a simple affair. Any mechanism that uses a constantly running clock needs a reliable power source. We could make use of an external synchronization signal to update the state of the timer, but that creates a vulnerability. Having the state of the clock saved in non-volatile RAM is also an option, but such a solution is both unwieldy and vulnerable to a freezing attack (where an attacker removes the tag’s power source to delay the expiration of the timer).

In [Sta02], Stajano describes a related weakness associated with using a watchdog timer in ubiquitous computing. The problem lies in the ability of a malicious application to simply keep telling the timer to give itself more time for execution. To combat this weakness, Stajano proposes a “grenade timer”, which will go off after a specific time, no matter what the conditions are. Stajano’s idea, based in the area of ubiquitous computing, does not however address how to ensure the timer is reliable.

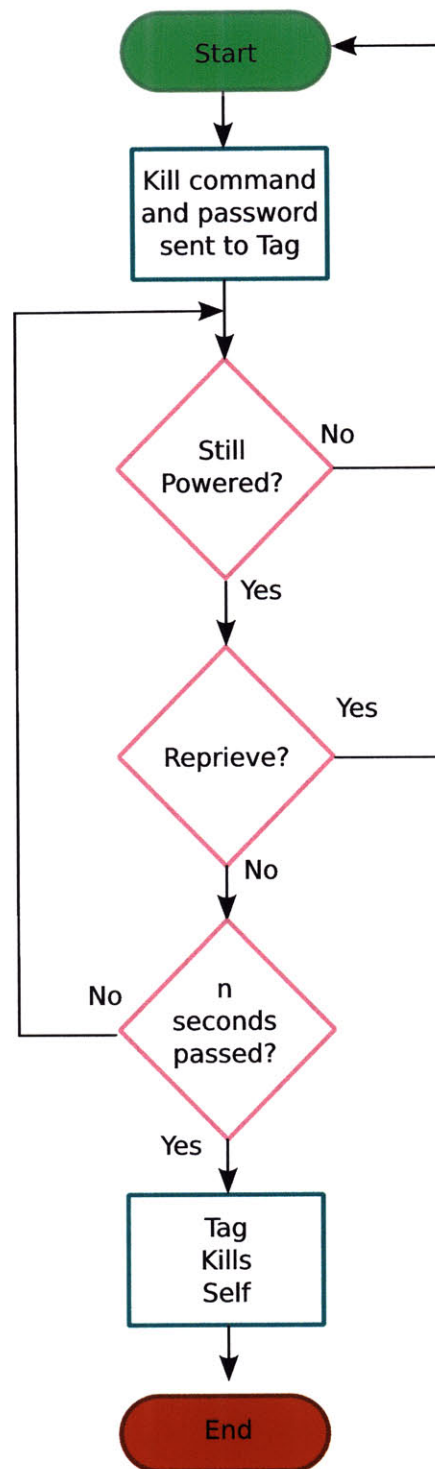


Figure 6-2: Flowchart of the Temporary Reprieve kill protection mechanism

To address our needs, I propose the concept of a “Deadline Timer”. To implement a Deadline Timer in our tag design, we first need to divide the tag’s RAM into physical classifications:

Random Access Memory (RAM) : stores values as long as power is present, can be written and read at any time.

Non-Volatile Random Access Memory (NVRAM) : stores values even if power is removed, can be written and read at any time.

One Time Programmable (OTP) : stores values even if power is removed, can be written once, but read any time.

Read-Only Memory (ROM) : stores values even if power is removed, can only be read – programmed by manufacturer.

In particular, I create a new subdivision of memory in the RAM and NVRAM classes:

Ephemeral: Memory that is valid for a specific time period enforced by its Deadline Timer. After that time period, the memory is explicitly erased.

Non-ephemeral: Memory that does not have an explicitly life-time or should be stored for long periods such as the seed for a random number generator.

If authorization information is placed in ephemeral memory, the lifetime of the authorization is restricted to the given period of time before expiration. After expiration, the same authorization credentials will no longer be accepted. For continued access, the previously authorized entity will need to request new tickets.

The above concepts work well when the tag has power, but become problematic to implement in a situation of unreliable power. I propose implementing the Deadline Timer using calibrated capacitors in the tag’s circuit to power the associated RAM. The capacitors will need to have sufficient capacity to refresh the RAM for the specified expiration time. Once the capacitors fall below a critical threshold, the memory will be erased. Starting and restarting the timer will require charging the capacitors. If the tag is taken out of a RF field sufficiently strong to power it, the capacitors will discharge until the timer expires. The calibrated capacitor concept thus creates a physically inviolate Deadline Timer, which we will need to make use of the following kill resistance mechanisms.

6.2.2.3 Stay of Execution

Recall that to outsmart the Temporary Reprieve mechanism, an attacker could use jamming to prevent the REPRIEVE command from ever reaching the tag. This attack would be easily detectable in an environment of readers programmed to expect it,

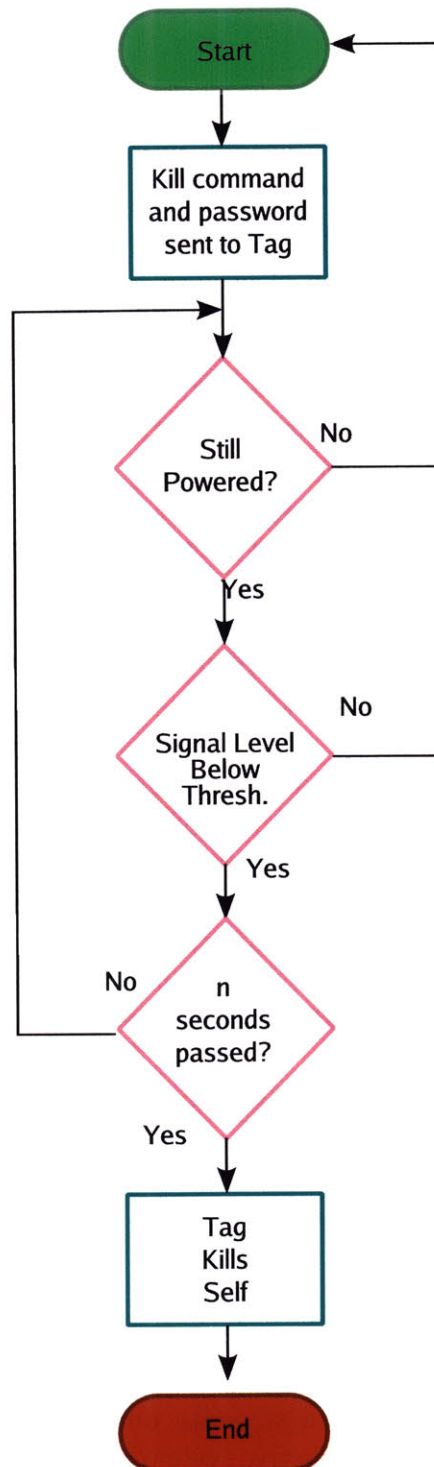


Figure 6-3: Flowchart of the Stay of Execution kill protection mechanism

but to combat it further, we can modify the kill sequence to include the expectation of a silent time period while the tag is dying. I call this concept “Stay of Execution”.

In the Stay of Execution system (illustrated in Figure 6-3), a tag will fire up its internal timer and begin listening for further signals upon receiving the kill command. If it hears nothing above a given threshold during the “Moment of Silence” (MOS) that follows, it completes its ritual suicide. If, on the other hand, it is not given proper respect and there is chatting, it stays alive and transmits a `ENORESPECT` signal.

A possible concern with the Stay of Execution model is that killing tags will become unreasonably difficult given the MOS requirement. However, to implement this system well merely requires constructing a properly RF-shielded area at the Kill kiosk, which will eliminate any outside RF interference. This could be as simple as a small grounded metal mesh enclosure around the reader.

Yet another concern with this model is the overhead needed to legitimately kill a large number of tags. The Temporary Reprieve mechanism called for waiting for a specified time period before the kill completed. However, during this time period, kill (or other) commands could be freely issued to other tags. The Stay of Execution model, on the other hand, requires complete silence for the tag to complete its execution. Thus, only one tag could be killed (or communicated with, for that matter) during the “waiting period”.

A small modification to the Stay of Execution protocol solves this problem. If we allow the kill command to specify a number of tags simultaneously, these tags can all complete their MOS at the same time. To use this system, we would need to query the tags first in order to get their IDs, then send the group Stay of Execution command. A friendly reader which noticed an unauthorized multiple-kill request would have the chance to interrupt it by sending a `REPRIEVE` or any other signal. In such a situation, the friendly reader may in fact wish to switch to jamming and alert the system for human intervention.

While the Stay of Execution model will succeed in protecting the tags from some unauthorized kills, it does suffer from the need to isolate the tags and reduce RF noise. The next system, Confirmed Kill, is an alternative designed to address the mass kill problem from a different angle.

6.2.2.4 Confirmed Kill

Ideally, we would like to have a system that allowed authorized readers to complete mass killing without the need for a separate authentication path and in an open environment. We can design a more active system capable of meeting this specification by requiring the kill command to include a confirmation signal before it completes its action. Such a concept befits the name “Confirmed Kill.”

The flowchart outlining the Confirmed Kill mechanism is presented in Figure 6-4. To give time limits to the kill authorization, we make use of a Deadline Timer, which

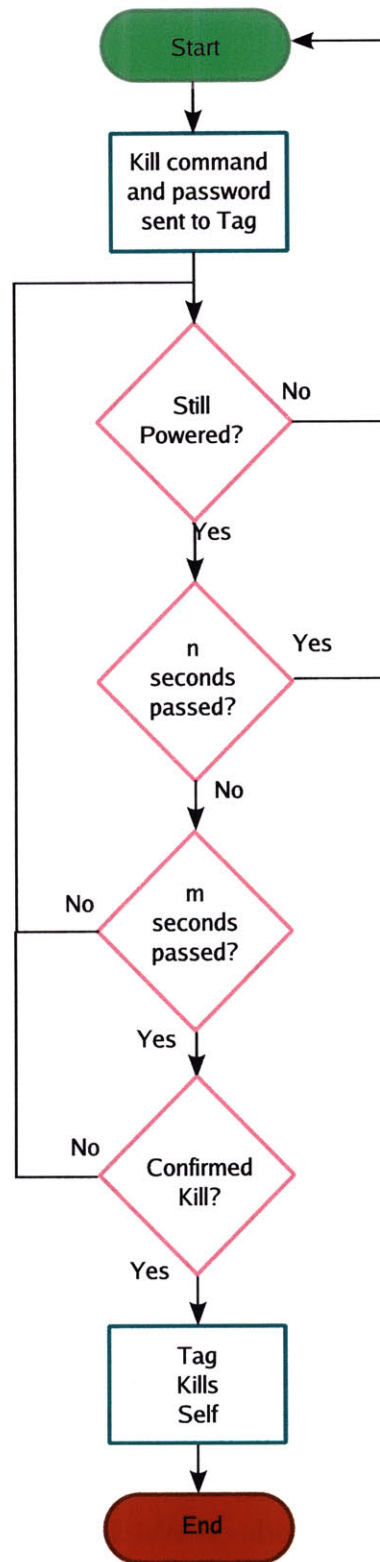


Figure 6-4: Flowchart of the Confirmed Kill protection mechanism

will reset the tag if n seconds have passed after receiving the kill command and the confirmation has not arrived. We also invoke a second timer that will not allow the acceptance of a CONFIRM signal until m seconds have passed since the original kill command. The operation sequence using Confirmed Kill would thus be:

1. The reader $\mathcal{R}$ asks tag $\mathcal{T}$ for its ID i .
2. $\mathcal{T}$ sends its i to $\mathcal{R}$.
3. $\mathcal{R}$ looks up the kill password (k_i) in the database.
4. $\mathcal{R}$ sends k_i and the KILL command to $\mathcal{T}$.
5. $\mathcal{T}$ verifies k_i and sends a random challenge c to $\mathcal{R}$.
6. $\mathcal{T}$ starts its timer m .
7. $\mathcal{R}$ computes $v = k_i \oplus c$.
8. $\mathcal{R}$ waits m seconds.
9. $\mathcal{R}$ sends v to confirm the kill.
10. $\mathcal{T}$ verifies v and, if v matches, kills itself.

However, if $\mathcal{T}$ receives an incorrect v or receives it too early, it resets c and begins again at step 1.

This protocol creates an intuitive mechanism for a protective reader to jam an unauthorized signal. The protective reader should be able to easily notice when an unauthorized reader tries to kill a tag at step 4. The reader can then immediately send an incorrect response to the tag to reset it, begin jamming or send the REPRIEVE command. Jamming should prevent this tag and others from being killed until a security agent can go investigate. Much like Stay of Execution, Confirmed Kill allows jamming to function in the favor of the protecting agent. However, in this mechanism, a reader can kill multiple tags without requiring an RF-isolated area. To allow an authorized user the ability to kill items, they should authenticate to the network via one of the methods discussed in Section 4.2.1.

A downside to any of the Neighborhood Watch solutions is that they only protect tags from mass kills in the presence of pro-active friendly readers. To protect tags in situations when such agents are unavailable, I designed ways in which tags can defend themselves against malicious attackers and will discuss these next.

6.2.3 Speed Bumps

To protect isolated tags from being killed rapidly, we need a systematic procedure for slowing down the killer. An analogy to this idea is the placement of speed bumps in densely populated areas. A traditional speed bump will not suffice in our situation, though, as we require killing of small numbers of tags to be quick, while execution of large numbers of tags to be difficult. What we therefore need is dynamic delay.

6.2.3.1 Puzzles

The mass kill problem holds similarities to the problem of spam currently afflicting the Internet. Both problems stem from the ease with which large numbers of electronic entities can be addressed. Not only does electronic addressing take little effort, it can be done very cheaply, eliminating the need to be selective. The Host Identification Protocol (HIP) ([MNJtH04]) and later [BS04] present interesting strategies for dealing with spam-infested networks, which inspired me to design similar solutions for our applications. Before I present my protocols, however, let's review the work that inspired me.

[BS04] proposes changing the email process to force a cost associated with sending email onto the client. The proposed cost, however, would not be monetary, but in the form of computer resources. The exact price of computing cycles varies heavily, but there is nevertheless an associated cost and delay with expending such resources. The cost proposed by both Beal *et al.* and Moskowitz *et al.* would come specifically in the form of reverse solving a hash, which requires finding a particular number that hashes to a desired result. The difficulty of this task can be varied by choosing how closely the output needs to match the desired result. We also call this mechanism "puzzle solving".

The HIP protocol, used to enable trusted communication between two machines which might have unstable network addresses, employs the puzzle solving mechanism during authentication exchange. The third packet passed during the exchange carries with it the puzzle, the goal of which is to find a padding bit-sequence that allows the entire packet to be hashed to a number where the last k bits are zero. This design allows the difficulty of the puzzle to be tuned by changing the value of k (since to solve the puzzle, the computer needs to perform an exhaustive search through a number field 2^k big). The difficulty can thus be varied from 0 (trivial) to 64 bits for a prohibitively hard search. The server need only expend minimal effort to verify the result.

In [BS04], Beal *et al.* discuss using the above mechanism to protect email from spam by forcing each email exchange to include solving a cryptographic puzzle cookie. Even though the resource cost is associated mostly with the client end of the e-mail transaction, the resource cost at the server end will also increase since the server must do more to process the email it has been handed. Unlike the HIP protocol, which is designed to be mostly stateless - the server does not need to remember that it

answered a request for connection, it merely checks that the solution it is given is valid,

I took the puzzle solving ideas and applied them to the RFID network to create a mechanism for protecting tags from unauthorized kills. I discuss the proposed mechanism in the following section.

Beal *et al.*'s implementation contains a look-up table on the server to further make sure that the solution has not already been used. This table and other details are used to prevent these attacks:

Counterfeiting: The attacker tries to send invalid cookies or incorrect puzzle solutions.

Time shifting: The attacker replays previous results he has stored up.

Collusion: The attacker shares solutions with other attackers.

Repetition: The attacker just keeps reusing the same solution.

I will also need to ensure that my solution is not easily circumvented using such attacks.

6.2.3.2 Exponential Rampup

The purpose of the puzzle mechanism we desire is to slow down killing. Given that, I propose that the tag have a dynamic solution requirement, but not store a look-up table of what solutions it has seen before. The reasoning behind this proposal comes from the fact that the kill transaction is expected to occur only once, in contrast to the e-mail server's need to deal with repeat requests. Once our tag is killed, we no longer care about its internal state because it will cease to answer requests.

I further propose designing two modes into the puzzle mechanism to be used based on the level of authorization of the kill command. A tag that receives a kill command from an authorized reader should return an easy puzzle. An unauthorized request, on the other hand, should trigger the return of a harder puzzle. This will make killing the tag more difficult, yet still possible. We would like to maintain this possibility for situations such as loss of passwords. In order to achieve the dual-mode feature, I designed a mechanism I call Exponential Rampup.

In the Exponential Rampup protocol, tags ($\mathcal{T}_0 \dots \mathcal{T}_n$) listen to each other, and the larger the community of the tags, the better protected they are. The protocol requires all tags to have a timer E , configurable by their owner. What follows is an example of a sequence of events that would take place if an unauthorized reader $\mathcal{R}'$ attempted to kill all n tags, starting with $i = 0$.

1. $\mathcal{R}'$ sends a KILL command to $\mathcal{T}_i$.

2. All tags that hear the KILL request (including $\mathcal{T}_i$) increment their puzzle difficulty registers d by 1. They also start their timers E .
3. $\mathcal{T}_i$ responds to the KILL request by returning a puzzle $Z()$ back to $\mathcal{R}'$. The puzzle requires reversing the hash of a random challenge c to match $d_0 + d$ bits (d_0 is the default puzzle difficulty).
4. If $\mathcal{T}_i$ receives back an incorrect answer to its puzzle, it aborts the kill sequence. To kill the tag, a new request (step 1) would need to be made.
5. If $\mathcal{R}'$ finds the correct puzzle solution ($a = H^{-1}(c)$) and returns (KILL, a) to $\mathcal{T}_i$, $\mathcal{T}_i$ will proceed to kill itself. Having succeeded, $\mathcal{R}'$ increments i , and begins again at step 1.

The purpose of the timer E is to reset the puzzle difficulty. Every time E expires, the tag will decrement its puzzle difficulty register d by 1 until d returns to zero.

The above protocol creates an exponential cost in resources within E associated with killing tags (Figure 6-5). If the time for a d_0 difficulty puzzle is $l \times d_0$ (where l is an arbitrary constant), then the time to kill (t_{k1}) a tag population of size n is:

$$t_{k1} = \sum_{i=1}^n l(d_0 + d) \times 2^i \quad (6.1)$$

In this equation, the final term quickly dominates, leading to:

$$t_{k1} \approx l(d_0 + d) \times 2^n \quad (6.2)$$

This system relies on the assumption that tags can hear readers telling other tags to die, which is safe to make as long as a large number of tags are in close proximity. An interesting side-effect of this protocol is the ability of a friendly reader to make a set of tags very difficult to kill. To do so, the reader simply needs to make a number of kill requests to the tags without solving the puzzles, all tags within earshot will increment their difficulty counters. If the counters end up being set unreasonably high, no tags can be killed for E amount of time. Furthermore, if a friendly reader hears an unauthorized KILL request, it can simply send a random “puzzle answer” back to the tag since incorrect puzzle answers reset the protocol. The friendly reader can thus save the tag’s life.

The disadvantages of the proposed mechanism are the additional requirements of a method for keeping track of time and the ability of the tag to do a cryptographic hash (for a discussion of hash function synthesis, refer to Section 4.4.3). We can use an alternate implementation of the Exponential Rampup protocol, which removes the cryptographic hash requirement, though it maintains the need to keep track of time. Specifically, the alternate implementation requires a configurable timer F_1 and two owner-defined timers E_1 and E_2 . F_1 defines the minimum time (t_{d0}) between

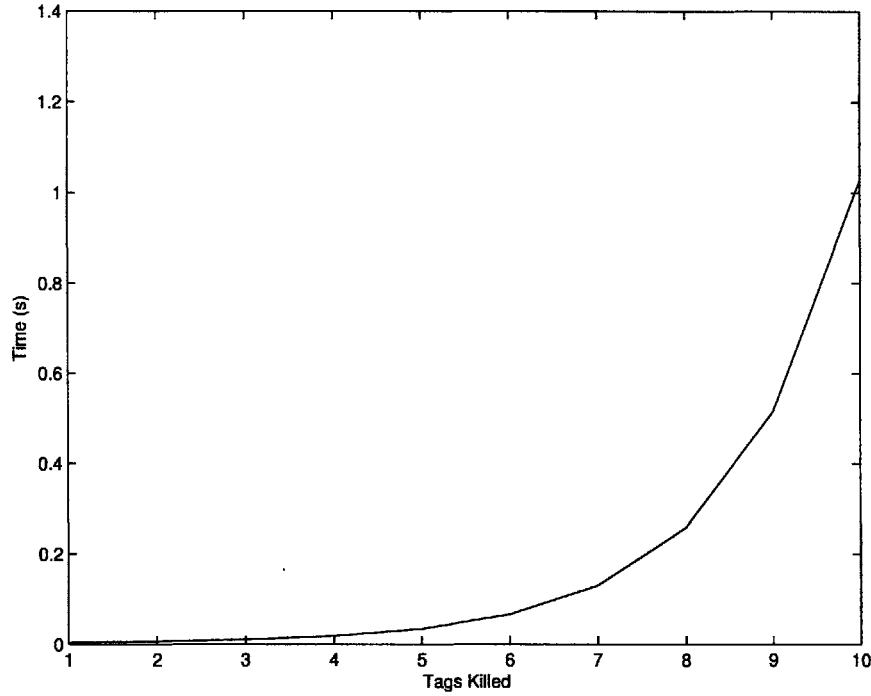


Figure 6-5: Simulation of Exponential Ramp-up kill resistance ($d_0 = 0.001$, $d = 0.001$)

the initial KILL request and the CONFIRM command. This time will be increased each time a tag witnesses a KILL request being made by the value of t_d , where $t_d = k \times 2^j$ (k is a constant and j is the delay increment). The E_1 timer resets the algorithm at the beginning, proving a timeout if the CONFIRM command is not received. E_2 decrements the delay increment j , analogous to the function of E in the first implementation. The same example as above (if an unauthorized reader R' wished to kill all n tags) would produce the following sequence of events:

1. The unauthorized reader R' gives a KILL command to $\mathcal{T}_i$.
2. All tags that hear the KILL request (including $\mathcal{T}_i$) increment their delay increments j by 1 and start their timers E_2 .
3. $\mathcal{T}_i$ sets its lifetime to $F_1 = t_{d0} + t_d = t_{d0} + k \times 2^j$. It starts F_1 and transmits its value to R' .
4. If $\mathcal{T}_i$ receives an ABORT at any time, the kill sequence will be aborted. To kill the tag, a new request (step 1) would need to be made.
5. After F_1 expires, $\mathcal{T}_i$ starts the count-down timer E_1 , which defines the window of opportunity to kill the tag.

6. If $\mathcal{T}_i$ receives a CONFIRM command during the window of opportunity, it will kill itself (a CONFIRM command at any other time is simply ignored). Having succeeded, R' increments i and starts again at step 1.
7. If E_1 expires and a CONFIRM command was not received, the kill is aborted.

This scheme is equivalent to the previous system in terms of resistance. The amount of time (t_{k2}) for a population of n tags to be killed is:

$$t_{k2} = \sum_{i=1}^n (t_{d0} + t_d) \times 2^i \quad (6.3)$$

$$t_{k2} \approx (t_{d0} + t_d) \times 2^n \quad (6.4)$$

One might expect a complication to this protocol might extend from the command interleaving or “busy signal” concept of Section 4.6.3. Command interleaving allows a long-running command to process on a tag while a reader interacts with another. An attacker might try to complete a mass kill by starting many kills sequentially, hoping to reduce the amount of time per kill. This is not the case in the puzzle-based scheme because the time limitation onus is placed on the reader. Killing only occurs as fast as the reader is able to solve the puzzles that have increasing difficulty each time a kill request is made.

The dynamics of the timer-based scheme change, but still make mass-kills difficult. With interleaving, the time (t'_{k2}) to kill a population of n tags is:

$$t'_{k2} = (t_{d0} + t_d) \times 2^n \quad (6.5)$$

Thus, t'_{k2} is equal to the term that t_{k2} approaches as n gets large (see equation (6.4)). This analysis indicates that interleaving will not reduce the effectiveness of the Exponential Rampup systems.

Considering these two schemes, we notice that they are resistant to the aforementioned attacks:

Counterfeiting: The puzzle-based scheme knows what the correct result is. There is nothing to counterfeit in the timer-based scheme.

Time shifting: The tag randomly generates a new challenge each time, making reuse implausible. There is nothing to replay in the timer-based scheme.

Collusion: The puzzle-based scheme can be attacked by increasing computing power. However, this does not decrease the cost, instead simply distributing the cost. In the timer scheme, there is nothing to share.

Repetition: As long as the tag does not reuse its puzzles, the attacker cannot reuse a given solution on other tags. There is no solution to repeat in the timer-based scheme.

Let us now switch gears and consider situations where we do wish to kill tags. We will examine how to properly authenticate entities to allow them to make protected kills.

6.3 Protected Kills

An entity capable of killing a tag should possess the following two characteristics:

Physical possession: The entity has physical access to the tag and its associated item. We call this requirement “Proof of Life.”

Electronic possession: An authority believes that the entity is the owner of that item and tag.

Electronic possession can be further subdivided into two categories:

Electronic owners: entities that have electronic rights assigned to them that cannot be taken away, only transferred

Electronic renters: entities that have been temporarily given rights to the object in question, usually in return for assets or resources

This differentiation is necessary to handle cases where access is given only to particular facets of an object. As an example, consider buying processing time on a supercomputer. The company is giving the user temporary access to a limited resource and will remove said access once the allotted amount of the resource has been utilized. Equipment such as cable de-scramblers and cable modems are often considered to fall into the same category – since the company that provides the cable service owns the box, a renter has limited rights and access.

Conveniently, some of these elements are already addressed by TagCheck and TagFolio. TagCheck can prove physical ownership of a tag, not just the ID it broadcasts. TagFolio can be used to authenticate to the network and perform the operations necessary to transfer ownership from one entity to another.

6.3.1 Proof of Life

Let us first consider the requirement of physical access to the tag, also known as “Proof of Life”. This is subtly different from the distance-bounding protocol needs. A distance-bounding protocol such as described in [HmGK05] increases the probability

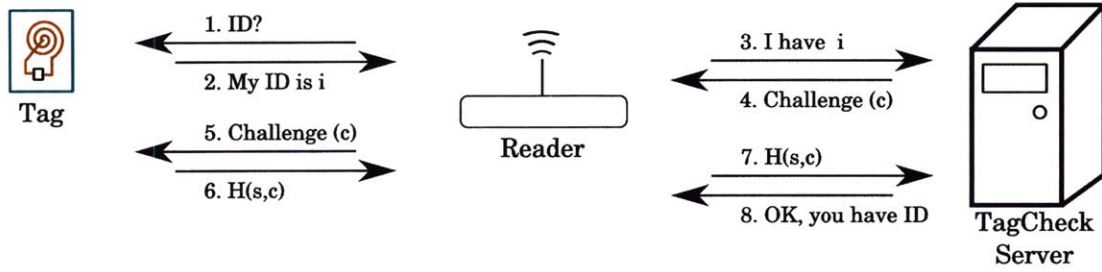


Figure 6-6: Proof of Life operation

that an entity is speaking directly with the tag. Our requirement is less; we need reassurance that we can communicate with the desired entity even if by proxy.

Conveniently, our requirement can be met using the following slight modification of TagCheck (also shown in Figure 6-6):

1. The reader $\mathcal{R}$ queries the field to find the tag $\mathcal{T}$'s ID i .
2. $\mathcal{R}$ tells the TagCheck server $\mathcal{S}$ that it has i .
3. $\mathcal{S}$ gives $\mathcal{R}$ a random challenge c .
4. $\mathcal{R}$ transmits c to $\mathcal{T}$.
5. $\mathcal{T}$ generates the response $H(s, c)$ using its internal secret s .
6. $\mathcal{R}$ forwards $H(s, c)$ to $\mathcal{S}$.
7. $\mathcal{S}$ now believes that $\mathcal{R}$ has seen $\mathcal{T}$ recently, and gives it a token specifying such.

Proof of Life allows an application to dispense with the need for secondary verification such as bar-codes. Furthermore, it is a crucial step to providing access control to the tag and we will use it as part of a composite service that gives authorization to a reader (or TagFolio) to complete tag operations such as unlocking, writing to, or killing.

6.3.2 TagDirective

With Proof of Life, we have taken care of ensuring that the entity wishing to modify a tag has physical possession of it. Now we need to address the issue of electronic possession, or ensuring that the entity should be allowed to complete the given transaction. This can be accomplished by modifying the TagCheck server to create a new service, which I call TagDirective. The capabilities of TagDirective are illustrated by the following example (and shown in Figure 6-7), in which we use TagFolio to complete a protected kill of a tag:

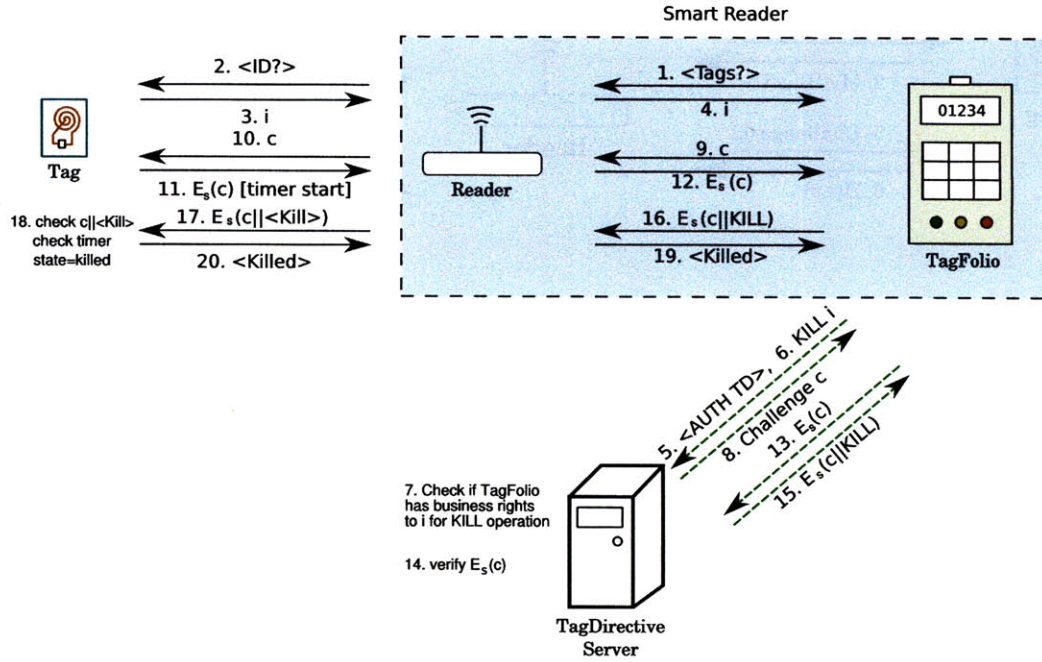


Figure 6-7: Diagram illustrating the operation of TagDirective

1. The TagFolio $\mathcal{F}$ gives a request to the reader $\mathcal{R}$ to inventory the tags.
2. $\mathcal{R}$ queries the tag $\mathcal{T}$.
3. $\mathcal{R}$ gets back the ID i from $\mathcal{T}$.
4. $\mathcal{R}$ transmits i to $\mathcal{F}$.
5. $\mathcal{F}$ decides that it wants to kill $\mathcal{T}_i$. It authenticates to the TagDirective server $\mathcal{K}$ using certificates and SSL.
6. $\mathcal{F}$ tells $\mathcal{K}$ it wishes to kill $\mathcal{T}_i$.
7. $\mathcal{K}$ checks its database (or queries another server) and determines that $\mathcal{F}$ has business ownership of $\mathcal{T}$.
8. $\mathcal{K}$ sends a random challenge c to $\mathcal{F}$, and stores it in non-static memory.
9. $\mathcal{F}$ sends c to $\mathcal{R}$ addressed for $\mathcal{T}_i$.
10. $\mathcal{R}$ sends c to $\mathcal{T}$ as part of a Proof Of Life request using TagCheck⁴.

⁴We choose the symmetric version for simplicity.

11. $\mathcal{T}$ generates $E_s(c)$ and returns it to $\mathcal{R}$. It also starts its Deadline Timer by setting $\tau = \tau_{end}$.
12. $\mathcal{R}$ sends $E_s(c)$ to $\mathcal{F}$.
13. $\mathcal{F}$ sends $E_s(c)$ to $\mathcal{K}$.
14. $\mathcal{K}$ verifies that $E_s(c)$ is correct.
15. $\mathcal{K}$ creates an authorization credential: $E_s(c||\text{KILL})$ and sends it to $\mathcal{F}$.
16. $\mathcal{F}$ forwards the credential to $\mathcal{R}$.
17. $\mathcal{R}$ sends the credential to $\mathcal{T}$.
18. $\mathcal{T}$ decrypts the credential using its key and notices that it says KILL. If c is the same as the one it sent out and τ hasn't reset the tag by reaching 0, $\mathcal{T}$ kills itself.
19. $\mathcal{T}$ indicates that the kill was successful (KILLED) as its final action to $\mathcal{R}$.
20. $\mathcal{R}$ forwards KILLED to $\mathcal{F}$.

The major advantage of the above system is that the tag's kill authorization can only be used for the short duration before the Deadline Timer runs out. Once the timer resets, the challenge memory is wiped and, consequently, the KILL request will not authenticate properly. In order to complete the kill, a new request would need to be made.

The possibility of abuse is further minimized because the reader and TagFolio are not responsible for storing the kill password. To prevent replay attacks, we could add a full nonce system, but this is unnecessary in the case of a KILL operation since it is only completed once during the lifetime of the tag.

An attacker might try to construct challenges to obtain the identity of the secret on the tag. However, choosing a strong cipher makes this attack mostly irrelevant. To further minimize such problems, we could use the exponential ramp-up mechanism in Section 6.2.3.2 to will quickly make multiple invalid queries unreasonable. To implement such a solution, a tag would double the amount of time between accepting TagDirective requests each time it received an invalid one. An attacker's attempts are then unreasonable in terms of time.

In the case of operations that might be used more than once⁵, such as updating the secrets or locking/unlocking a tag, the same system will work but more care will need to be taken to ensure that the possibility of a challenge being used more than once is minimized. Assuming a good pseudo-number random generation system is used, this requires picking a good initial seed value and keeping state.

⁵or in the case of a single kill password being used for many different tags

The system also requires network access for the reader or TagFolio to authorize its request. If disconnected full-speed operation is required, it will be necessary for the secret to also be stored somewhere else. If the secret were on the inside of a packaging lid or flap such that tampering with it were evident, then the kill station could also function as the authorization server. With knowledge of s , it could easily produce $E_s(c||\text{KILL})$.

6.4 Summary

In this chapter we discussed the complexities of access control and management when applied to RFID tags. I introduced the concepts of physical and electronic possession and examined ways to prove such ownership, needed to perform authorized operations on tags. To prove physical proximity (equivalent to physical possession), I designed the Proof of Life protocol. I proceeded to use the protocol as part of a new system, TagDirective, designed to authenticate entities to perform key tag manipulations. TagDirective provides a time-limited access control scheme and, much like the other mechanisms in this thesis, eliminates secret exposure to vulnerable elements such as readers.

In contrast to allowing authorized operations, I described mechanisms to prevent malicious mass killing of tags. I developed the concept of Neighborhood Watch, creating a community of friendly readers to protect tags from mass kill attacks, and Exponential Rampup, to allow the tags to resist such attacks even when isolated from friendly readers. All in all, the given solutions make mass killing time-consuming and easily detected.

In the next chapter, I examine information leakage at a higher level. The data encoded in the ID of an RFID tag may provide additional information to an attacker when read in isolation or during resolution. I explore the ramifications of the existence of such information and how to prevent its misuse.

Chapter 7

ID Resolution on the Network

“Your name was Raymond K. Hessel on your driver’s license. You live at 1320 SE Benning, apartment A. That had to be a basement apartment. They usually give basement apartments letters instead of numbers.”[Pal96]

Up to this point, we have discussed tag and reader security. We now turn to the network over which the devices communicate with each other. The network itself can be the weakest link in the chain in terms of security, but I present solutions for how this problem can be addressed.

In EPC architecture, information can be gleaned from the ID itself. Even if all the security solutions I presented in the previous chapters are used, an attacker can still obtain a given tag’s ID when it’s passed between the tag, the reader, TagFolio, or the TagDirective server. They will not be able to gain access to the tag, but given the manufacturer and product type (assuming these are encoded in the ID)¹, they could gain information about that product without even querying the EPC-IS server. Thus, someone could still find out what medication our friend Bob is taking to treat his embarrassing illness. This attack is equivalent to collecting information from the cookies used by web browsers.

There are two obvious ways to deal with this issue while still using the EPC architecture. The first one, temporarily silencing or obscuring the tags, can be used to mitigate the problem by choosing when the tag’s ID is exposed. However, this is not truly a solution since it doesn’t address the problem itself. What we would like to do instead is to find a way for ONS to hide information from unauthorized entities.

In this chapter I will examine both these solutions. I will show how tags can be efficiently silenced and how to use anonymization to reduce information leakage during ONS resolution. Before discussing the solutions, however, let us examine the attacks that make this protection necessary in more detail.

¹If the ID is randomly generated or encodes no such information, this attack is irrelevant.

7.1 ONS Man-in-the-Middle (MITM) Attacks

As discussed in detail in Section 3.4.2, a common type of attack against many communications protocols is the Man-In-The-Middle (MITM) attack. To illustrate this attack, let's consider Alice and Bob, who wish to communicate securely using public key cryptography [Sch96]. The steps needed to complete their communication are as follows:

1. Alice sends Bob her public key.
2. Bob sends Alice his public key.
3. Alice encrypts her message with Bob's public key and sends it to Bob.
4. Bob decrypts the message with his private key and reads it.
5. Bob encrypts his return message with Alice's public key and sends it to Alice.
6. Alice decrypts Bob's message with her private key and reads it.

This simple exchange may seem secure, but it is very vulnerable to a MITM attack. Consider Mallory, a malicious attacker, who would like to spy on Bob and Alice's conversation:

1. Alice sends Bob her public key. Mallory intercepts Alice's key, and substitutes his own public key.
2. Bob sends Alice his public key. Mallory again intercepts Bob's key, and substitutes his own public key.
3. When Alice sends a message to Bob (encrypted with what she believes to be Bob's public key, but is in fact Mallory's public key), Mallory intercepts the message and decrypts it using his private key. He then encrypts the message using Bob's public key and sends it to Bob.
4. When Bob returns a message to Alice (encrypted with what he believes to be Alice's public key, but again, it is actually Mallory's public key), Mallory intercepts the message and decrypts it using his private key. He proceeds to encrypt the message using Alice's public key and sends it on to her.

Mallory is thus able to read as well as modify any messages exchanged between Bob and Alice. This is a problem in any situation where data integrity is important. In particular, in the case of ONS using public cryptography, this is of serious concern since the information is often linked to items of value.

The easiest way to prevent MITM attacks, and the most recommended, is to use DNSSEC's authentication mechanism, which prevents DNS clients from being

hijacked. For full details on how DNSSEC works, refer to Section 2.3.8. In particular, DNSSEC focuses on preventing DNS poisoning attacks, in which a misconfigured server is convinced that incorrect records are authoritative. Since DNS information is cached, it can spread to any server or client that uses the misconfigured server, thus spreading the poison. However, the signature and authentication chain of DNSSEC can quickly detect and remove from the cache any non-authoritative data that does not have the proper signatures and consequently fix the problem. Unfortunately, DNSSEC does not obscure information, making it insufficient for our needs.

7.2 Silencing/Obscuring the Tags

The obvious way to prevent someone from gaining information (such as a tag's ID) when it is passed between the tag and the reader is to force the tag to keep quiet. An effective and inexpensive way to do this is to put the tags inside a metal box. This "Faraday cage" keeps any RF from reaching the tag to query it. However, we will not be able to communicate with the tag in this configuration. Destroying the tag also works for silencing it, but at the expense of subsequent use. For example, one could design an external tag that can be ripped to silence it ([Baa06]), but this will destroy the tag at the same time.

Additional methods for temporarily silencing tags are detailed in Section 2.1.9.3. TagFolio is also able to achieve this by reprogramming a tag with the SILENCE command (see Section 5.1).

Silencing tags works well in situations when we don't wish to communicate with the tag. But we also desire to maintain the same level of security while the tag is actively communicating with the reader. Certain tag systems, however, make eavesdropping easy. A RFID tag reader that uses the binary tree-walking algorithm² is effectively broadcasting the tag's ID. An eavesdropper simply needs to listen to the reader's queries to gain the information they need to reconstruct a given tag's ID. They don't even need to hear the tag's response to the reader, which is troubling because it is physically more difficult to listen to a tag than a reader due to the differences in signal level (the differences are several orders of magnitude).

[DO05] describes one way of making the eavesdropping problem described above more difficult. Instead of using its own ID to do the anti-collision, the tag chooses a random number when powered up. This number becomes its anti-collision ID for the same anti-collision algorithm. Once anti-collision negotiation is complete, the tag can transmit its real ID. The tag could even continue using its anti-collision ID as its real ID later to make tracking more difficult, but this highly limits the usefulness of the tags for anything other than "counting the cows" (counting how many tags are present).

The JanusTag concept of rewritable public identity, which I developed earlier in

²Tree walking is explained in detail in Section 2.1.8.

this work, can also be used to obscure a tag's identity. Refer to Section 4.1 for further discussion.

All the above mechanisms protect the ID at the source, but a user will need to resolve the ID at some point in order to make use of its associated information. At that point, an attack on the resolution system becomes possible. Protecting against such attacks is the subject of the next section.

7.3 ONS Anonymization

To prevent an attacker from tracing the source of a particular ONS query, we require anonymization. The current line of anonymization research aims to explore the concept of disassociating a product's ownership from the manufacturer so that the manufacturer cannot trace where it is and who has it. One proposed method to achieve this is to create a scrambled space in the ONS name-space that points at many different services. A secure proxy would know the correct mapping between the anonymized tag and its actual EPC-IS server. However, this would make it very hard for someone to get information about the product without making a direct request to the EPC-IS server, which can then be discriminated by other access control mechanisms. Consequently, the administration of that space would be difficult making the application design hard.

To assist in anonymization, ONS proxies should be placed to concentrate network topologies. For instance, all of the ONS traffic for an apartment building might go through one proxy such that the finest level of granularity is still broad. For added security, a system of mix-nets could be used to make this even harder [FH01]³. The convoluted routing provided by mix-nets would make it very hard to determine where given packets came from.

7.3.1 An Anonymizing Proxy: The Onion Router (Tor)

One method using anonymizing proxies that was further investigated and implemented made use of a service called "The Onion Router" (or Tor) developed by Mathewson and Dingledine *et al.* [DMS04]. Tor is a cryptographically protected anonymizing proxy system focused on web applications. It is a generalized SOCKS4a TCP anonymizing proxy designed to make government-level anonymization possible and yet accessible to an average end-user.

Tor uses three intermediate proxy servers with encrypted links that make discerning the originating host difficult unless a significant number of the proxy servers have been compromised. Unfortunately, since the service creates virtual TCP circuits that maintain state and pass packets with minimal delay. Theoretically, this constraint

³A mix-net is a large network of servers that take packets and randomly (or pseudo-randomly in some cases such that a return path is possible) routes them through the network.

might allow limited traffic analysis with an attacker that could determine the entry and exit point. In practical usage with the numerous exit nodes, this attack is impractical.

Thanks to the anonymizing nature of Tor, one user was able to figure out that his company's competitor was playing ugly tricks on them. He noticed that when accessing a particular website, he got different results when using Tor than when accessing it directly from his office computer. Apparently, the competitor figured out that a given IP address set was only coming from the user's company, and set up their web-server to provide modified content pages to these addresses. In today's cutthroat markets, tricks like these are quickly becoming typical as competing corporations try to stay ahead of each other at all costs.

Let's take a look at how Tor actually works by considering a scenario in which Alice wishes to find a path to Bob (refer to Figures 7-1, 7-2).

1. Alice connects securely using public key cryptography to a directory server.
2. The directory server returns the public keys of three different Tor proxy servers (also known as "nodes").
3. Alice's client generates an Onion routing packet with information about each link in the route encrypted by the public key used by the given node.
4. The client sends the Onion routing packet to the first node.
5. The first node decrypts the first layer of the Onion routing packet, which contains the IP address and public key of the second node.
6. The first node forwards the partially decrypted Onion routing packet to the second node.
7. The second node decrypts the next layer of the Onion routing packet, which contains the IP address and public key of the third node.
8. The second node forwards the partially decrypted Onion routing packet to the third node.
9. The third node decrypts the last layer of the Onion routing packet, which contains Bob's IP address. The packet is now completely decrypted and ready to be sent to Bob. It will appear to come from the third routing node, not from Alice. The lack of encryption on the last path allows Tor to be used with any TCP-based protocol without the need to make changes to the application's server.
10. Bob can now make use of the "virtual circuit" by replying to the last Onion node as though it was Alice. The node will encrypt the packet and send it back

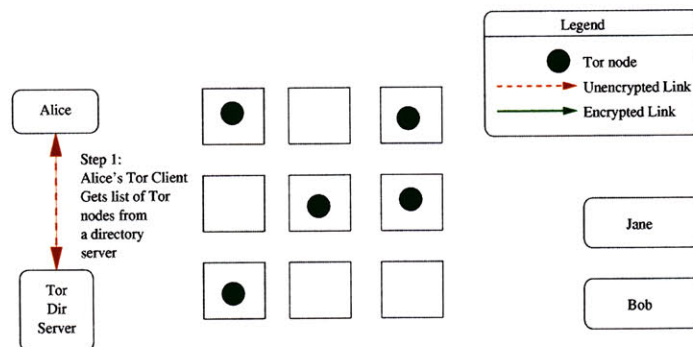


Figure 7-1: Step 1: Getting a list of Tor nodes

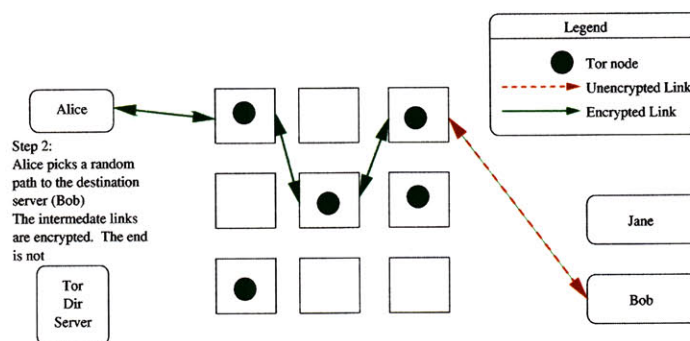


Figure 7-2: Step 2: Setting up a virtual circuit

via the established path to Alice. Each time the packet arrives at a node, it is decrypted with that node's key and encrypted with the next node's key. The final node is Alice, who can read the packet.

7.3.2 TorONS

Tor allows one to make anonymized ONS queries using an included DNS client called `tor-resolve`. Unfortunately, each query requires setting up a new virtual circuit, which takes on the order of 10–30 seconds. For the security-conscious user, `tor-resolve` provides a slow but secure solution because it ensures that all queries are made from different end-points. As a result, building up information about who made what queries is difficult. Some information can still be obtained if the queries themselves contain it, but addressing this concern is beyond the scope of my thesis.

In the context of RFID, however, the rate at which `tor-resolve` worked was insufficient. The limitations of the client inspired me to write an improved interface to allow fast yet secure ONS lookups. Using the Tor specification and software, I redesigned the Java network functions to make use of Tor anonymization for the

purposes of ONS. The majority of this work required modifying the native Java network functions to encapsulate the SOCKS4a proxy initialization. The modified functions were packaged into a Java Application Programming Interface (API) called TorLib. Making use of these modified SOCKS4a-capable Java network functions, I redesigned the resolver functions of the DNS-Java package by Wellington [Wel] to use TorLib⁴.

To test the capabilities of my completed client (called TorONS) against `tor-resolve`, I needed a set of test hostnames to represent ONS lookups. I produced a list of 100 hostnames by doing a google search on the word “Tor”. I then wrote a perl (for `tor-resolve`) and Java (for TorONS) timing program that had each system process the 100 host names and record the amount of time it took. The results of the tests are shown in Table 7.1.

Anonymized ONS Interface	Trial 1	Trial 2	Trial 3
<code>tor-resolve</code> 0.1.0.17	3,000	N/A	N/A
<code>tor-resolve</code> 0.1.1.24	159	31	0.5
TorONS	230	41	40

Table 7.1: Benchmarks (in seconds) of `tor-resolve` and TorONS

It should be noted that I completed the final version of the TorONS client in June 2006, a year after the initial test with `tor-resolve` (0.1.0.17). At that time, a new version of `tor-resolve` (0.1.1.24) had come out, which had been redesigned to solve many of the sluggish DNS lookup problems that had plagued the previous version. It was able to outperform TorONS because executed DNS entries were now cached on the local DNS client. As a result, additional lookups were extremely fast, as previously described. TorONS could certainly be further optimized to compare to the current `tor-resolve` version, but this was not the focus of my research. The goal, which I accomplished, was to prove that Tor could be used at all to reasonably protect ONS look-ups.

7.4 ONS Misdirection

The idea, proposed in Section 6.2.1, of emulating non-existent tags to draw out an attacker and obscure real tags from a monitoring agent can be taken in another direction to address the ONS security issues. Since potential exposure of information occurs when an agent accesses the ONS service, we can make fake ONS queries for non-existent items.

⁴This would not have been necessary except that the DNS API in Java is not designed to communicate to a non-standard port and with a SOCKS4a initialization sequence.

More formally, consider an agent $\mathcal{A}$ in possession of a number of tags $\mathcal{T}_0..\mathcal{T}_n$. $\mathcal{A}$ would like to find out what EPC-IS is associated with $\mathcal{T}_i$ without exposing such a request to a monitoring agent.

Using ONS Misdirection, $\mathcal{A}$ would make a look-up in the following way:

1. $\mathcal{A}$ constructs f false tags $\mathcal{T}'_0..\mathcal{T}'_f$ from a combination of previous queries and a local database.
2. $\mathcal{A}$ shuffles $\mathcal{T}_i$ into the false tags.
3. $\mathcal{A}$ makes ONS queries of the shuffled set of tags.

Following such a procedure makes determining which query is the one of interest difficult. However, constructing the fake tag information is non-trivial.

7.4.1 Construction of Fake IDs

A simple method of fake tag construction would be to pick random serial numbers and random manufacturers. However, this method is insufficient due to a sparse tag space – the ID with a valid ONS entry would be the tag of interest.

Another way to construct fake IDs would be to reuse IDs that the application has had in its possession before. This has the advantage of all IDs being valid, but it may expose even more information through providing past history.

Optimally, fake IDs would be endorsed by administrators of the various ID namespaces. A person wishing to make use of fake IDs could request unallocated IDs (or a reserved number range) from the administrator. Once they were done with the fake IDs, they would release them in much the same way Dynamic Host Configuration Protocol (DHCP) gives and releases leases for IP addresses. Such a system would require a significant amount of organization, but would allow fake IDs to be effectively indistinguishable from real IDs.

7.4.2 Using Fake IDs to Detect Attackers

It is possible to use the ONS resolution system in order to detect attackers. To demonstrate this, I will draw upon my previous research, GlueSniffer, that I implemented while doing software development for MIT's Information Systems (IS) Network Security Group (NetSec) [Fol97].

In 1997, MIT's network was having a great deal of trouble with “script kiddies”⁵, who would break into MIT UNIX machines and set up packet sniffers to gather passwords for other computers. At the time, encrypted logins were not required and, as a result, the sniffers were able to gather a significant number of passwords. Once

⁵slang for young individuals who use computer exploit software without understanding how it works

some passwords were gathered, they could be used to log in to other machines and set up more sniffers. Detecting such intrusions often took significant amount of time and effort, and the sniffers were not usually discovered until after the intruder had already gathered sufficient information for their next attack. As an MIT Network Security employee, I began researching sniffer software and found a way to discover sniffers much faster using a common feature – DNS resolution.

I began by constructing a DNS sub-domain in MIT's DNS structure called `lcc.mit.edu`⁶. There was only one valid host, `glue`, in the `lcc.mit.edu` domain. It ran a program that randomly made network connections to selected IS machines on MIT's sub-nets where we suspected packet sniffers were being used to gather passwords. When a packet sniffer with DNS enabled noticed `glue`'s packet go by, the DNS resolver would look up the A record for that domain. When the resolver got to the final DNS server in the chain it happened to be `glue` itself. When the DNS server on `glue` got this DNS request, it compared the requester against a list of hosts performing the test. If the requester was not on the list, it was added to a list of "sniffing" hosts. This service allowed us to detect sniffers by making use of their DNS lookups on packets that only they would notice.

A similar service could be performed for RFID tags using almost the same architecture. To implement this, parts of the ONS space would be reserved for attack detection. The tags that lived in such spaces could be considered sticky, like a fly-trap, and the ONS records for the sticky tags would point at a GlueSniffer-like server. An RFID-enabled application would use a separate secure channel to request a sticky tag from a server. The reader would then emulate a read request for a sticky tag, after which the application would perform an ONS resolution of the sticky tag, just like it would for a tag that existed.

Following the completion of the request, the reader would query the GlueSniffer to determine if anyone else had performed ONS resolution. Other agents making requests would indicate that a monitoring agent was within read range of the reader. False positives are highly unlikely since the tag is defined to not exist!

Another interesting idea to consider is programming a tag with a sticky tag's ID. Doing so would allow a consumer to gain some information about what monitoring agents have access to the readers in the tag's proximity. Such a service would satisfy those who believe that a person should know when someone is making use of information on their person.

A caveat for ONS Misdirection is usage places additional loading on the communication mediums. The extra traffic created will be inversely proportional to the Signal-to-Noise Ratio (SNR) of fake/real ID information. The amount of fake information used will therefore vary depending on both hardware capabilities and the specific amount of false information needed to hide valid information. The latter is currently an open question in the field of steganography.

⁶which stood for "Laboratory for Catching Crackers"

7.5 Security Boundaries

A concept that shows up in many security contexts is that of a boundary, often likened to a castle wall [BF04]. The purpose of the boundary is to prevent unauthorized information from entering or leaving. Breaches of the boundary or errors in the placement of sensitive information can have dire consequences. For example, in 2000, at UK's Powergen corporation, a "technical error" placed a file containing sensitive information outside the website's "security gate" or boundary. The information included names, addresses, and credit card details of customers who used the website to pay their bills. Major security breaches of this sort are estimated to occur at a frequency of about once a week, at an estimated cost of 120,000 pounds per intrusion [Man06].

A traditional way to create a security boundary is to deploy a firewall. A firewall is a piece of hardware or software that sits between two segments of a network. The firewall looks at a given segment and, using a set of rules, figures out if a particular data packet should be allowed through. Some firewalls not only filter, but also rewrite packets to new specifications (*e.g.* Network Address Translation (NAT)).

Properly set up, a firewall makes it much harder for a malicious entity to get at the hosts watched over by the firewall. Unfortunately, a naive traveling user who is using an outside connection will not be able to access machines inside the firewall, since, to the firewall, the user will be indistinguishable from an attacker.

The traditional way to allow users to access the "private network" created by the firewall is to set up a specialized proxy server behind the firewall that relays data for the user that is outside. This is called a Virtual Private Network (VPN). It is "virtual" because there is essentially a remote island in a separate location virtually connected to the inside of the private network.

For application within RFID, this concept of a boundary is encapsulated in the previously mentioned demesne system described in Section 2.3.4.3. The demesnes system uses the user's location, internal or external, to create the boundary. A tag's ID designed to work everywhere will have an internal and external presence⁷. An internal-only ID will only work while the user is inside the private network or making use of a VPN. Looking up the ID while outside of the network will not return a result. Such a boundary makes security easier by encoding the constraint directly in the ID itself.

7.6 Summary

In this chapter, I analyzed the insecurities that result in leakage of EPC data via the network. Most importantly, it is imperative to protect information associated with a given tag's ID if the ID encodes useful information.

⁷which may give the same result, or it may not.

A simple method for doing so is to silence the tag either temporarily or permanently, or to change its public presence as in JanusTag. An alternate method, which can be used in parallel, is to protect the resolution phase from being monitored. To this end, I designed TorONS, which protects ONS resolution by using the Tor anonymizing network to prevent traffic analysis. I also discussed the idea of ONS Misdirection, in which non-existent tags are emulated to obscure real tags from monitoring agents. ONS Misdirection can even be used to detect whether monitoring agents are present.

This chapter concludes the discussion of individual components I designed to secure the various elements of the RFID infrastructure. The next chapter combines all the components, creating a complete RFID protection system, which I call TinFoil.

Chapter 8

System View

“When Tyler invented Project Mayhem, Tyler said the goal of Project Mayhem had nothing to do with other people. Tyler didn’t care if other people got hurt or not. The goal was to teach each man in the project that he had the power to control history. We, each of us, can take control of the world.”[Pal96]

With the systems developed in this thesis, we have essentially stocked our toolbox full of tools needed to tackle RFID security challenges. To paint a complete picture, let us combine the tools into a single system that will address the functional requirements specified in Chapter 3. Before we begin, though, let us first review the individual tools and the industries that may be interested to use them.

8.1 Tool Review

8.1.1 JanusTag

The JanusTag has the ability to switch between private and public identities as needed. Privacy activists and the military are the groups most interested in such a tag. The military might use it to place a tracking number (PID) on containers that can be recoded as the containers move around, but still retain the permanent RID for internal tracking purposes.

Other areas that will likely be interested in recoding as a manner of privacy protection include libraries, video rentals, and any other service that loans or rents items¹. The renter will be able to silence the tag (or set it to a secured mode) by shutting off the PID. As a result, the tag will not respond to requests from public readers. However, the library, who, as the owner, knows the item’s RID, will still be able to communicate with the tag and therefore re-index the item when it is returned.

¹This is especially true for items that are potentially compromising, such as adult books and videos.

8.1.2 TagCheck

TagCheck, the anti-counterfeit mechanism for RFID tags that does not assume a shared secret with a reader, will be of special interest to the medical and pharmaceutical industries. Both of these areas are concerned with proving that items are genuine since counterfeits cut heavily into their profits and, more importantly, expose them to liability. The areas of gaming chips, RF-enabled smart credit cards, and money also have need for proof of authenticity, which TagCheck will enable.

8.1.3 TagFolio

The shipping industry already has the service model in place to make use of TagFolio: items come in through gateways and are transferred from one owner to the next, each of which has a separate database. The hash lock tag modification system addresses ownership transfer and application of personal security policies for future applications.

Near-Field Communications (NFC) financial transaction services also need the ability for leakage-free ownership and secret transference. Use of an NFC device to buy food at McDonald's should not allow an attacker to buy a plasma screen TV at Best Buy. In the current model, such an attack is possible, demonstrating the need for improved protocols. In the near future, the need for a separate TagFolio device will likely disappear as RFID-enabled phones and PDAs will run the TagFolio software directly.

8.1.4 Kill Resistance

In order to satisfy the needs of the privacy fundamentalists and the manufacturers who cater to them, RFID tags need to include a kill feature. It is important to recognize, however, that the presence of the kill feature opens avenues of attack by malicious attackers. Unauthorized mass killing will wreak havoc in the warehouses since vendors rely heavily on supply chain information provided by the tags. To protect the vendors, while also satisfying the desires of the Privacy Fundamentalists by keeping the ability to kill tags, we have developed several strategies that hamper tag mass killing.

8.1.4.1 Neighborhood Watch

This strategy uses the kill token as a warning mechanism to protective readers. The concept of a collection of readers is already inherent in the EPC savant model, as is a distributed event model. However, the savant model is focused on providing a virtual database centered on EPC events, not on actions to apply. Neighborhood watch would fit well as an additional module for the savant system.

8.1.4.2 Deadline Timer

The Deadline Timer is an inviolate timer that controls ephemeral memory, limiting access token lifetime. A similar implementation already exists to enable the sessions anti-collision method used in Gen2 [EPC05a]. The novel application of a discharging capacitor as a hard memory reset will be most interesting to high-security applications. The alternative method of capacitor value checking upon initialization may be vulnerable to carefully modulated power levels.

8.1.4.3 Stay of Execution and Confirmed Kill

These neighborhood-enabled kill resistance methods interpret any RF activity as an abort (to reduce the effectiveness of jamming) and provide a window for a friendly reader to jam, respectively. To work, they both rely on the presence of protective readers within the read range of the tags. As such, groups that have reader-dense inventory systems will be the main users of these kill resistance methods. Examples of such users include retail chains, warehouses, and shipping services.

8.1.4.4 Exponential Rampup

Exponential Rampup provides a kill rate limiting mechanism that uses the broadcast nature of RF transactions to make the tag killing operation non-linear in duration. Since this solution does not rely on a protective reader neighborhood, it is equally effective in isolated and small environments. Areas that are not as dense in reader coverage out of necessity or cost will most benefit from such measures. These include small retailers, convenience stores, shipping and postal transfer points, and container trucks.

8.1.5 TagDirective

Companies that loan or rent items will benefit greatly from TagDirective, the access control mechanism that gives time-limited tokens for a reader and TagFolio to use during tag operations. With TagDirective, items can be assigned temporary owners and, once the allotted time expires, give information for their return. Libraries, video rentals, and tool rentals are the obvious environments where a system of temporary access privileges will be beneficial. A more time-critical example is medical record authorization. A medication bottle may not present its RID without authorization from its owner, unless there is a medical authorization via TagDirective.

8.1.6 Reader Misdirection

Reader Misdirection provides a method for readers to fool eavesdroppers into thinking that non-existent tags are present. It will be most useful in environments where

monitoring malicious agents are likely to appear. The military could use Reader Misdirection to augment their RFID systems and prevent long-distance tracking. Gaming chip manufacturers might find the active misdirection a good tool to find certain counterfeiting efforts. The invalid numbers provided by these readers might be picked up by a skimmer and copied onto their counterfeit chips. In such a case, detecting and tracking a counterfeiter becomes trivial.

8.1.7 TorONS

Unlike the tools described so far, TorONS, the anonymized service that makes traffic analysis methods for eavesdroppers on EPCs difficult, has already been implemented². Privacy fundamentalists, government officials and tech-savvy politicians are the individuals likely to use TorONS the most when interacting with the EPC network. Corporations making large numbers of ONS requests as part of their track and trace networks will follow suit if their competitors begin monitoring such requests as part of data mining efforts. With a widespread use of TorONS, a consortium will likely be formed to host and distribute the large number of Tor servers needed to handle the necessary bandwidth.

8.2 TinFoil: The Complete System

I call the combined system that encompasses all the above tools TinFoil, a name inspired by the metal-lined booster bags referred to in Section 2.1.9.3. The innovations covered under the TinFoil umbrella could also be added to the Gen2 architecture as shown in Section 4.7, but I choose instead to create a new architecture to subsume Gen2. The specific areas of the EPC architecture where information leakage is minimized and the tools used to minimize the leakage are detailed in Figure 8-1.

To take full advantage of TinFoil, we will need to slightly modify the individual tools to take advantage of overlapping capabilities. The modifications necessary for optimal tool integration are outlined in the next section.

8.2.1 Integration of Individual Elements within the TinFoil Package

In order to smoothly integrate the individual sub-components of TinFoil, only a slight modification to one of its elements, TagFolio, is required. The modification involves adding the ability to use TagCheck authentication to $L_2()$'s capabilities, and the ability to change any of the tag's writable areas to $L_3()$'s functions. Neighborhood Watch requires TagDirective to have a well-known public key that can be used for a signature mechanism, but this does not impose any significant additional requirements

²see Appendix B.1 for details

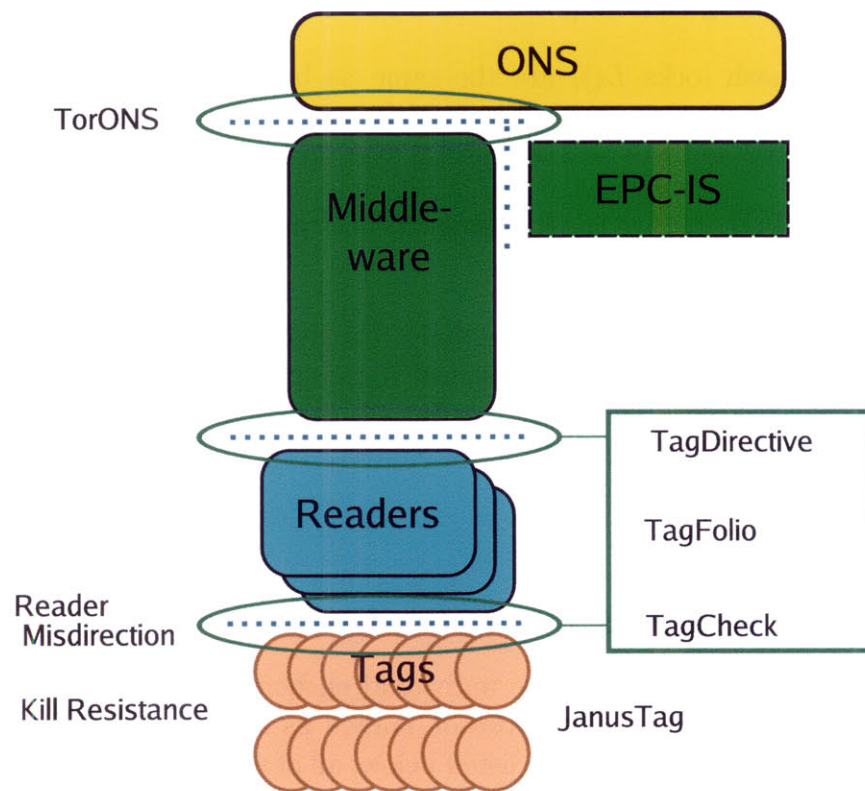


Figure 8-1: Interfaces where TinFoil elements reduce secret leakage

upon the tag. Readers that are protecting TagDirective tags, however, need to be able to perform signature verification.

The capabilities and memory structure of a fully integrated TinFoil JanusTag are shown in Table 8.1. The notation used across the individual elements within TinFoil as given in the table is summarized here:

TagCheck: TagCheck needs a secret s for authentication. We will denote this 128-bit secret s_s and place it within the One Time Programmable memory. We will also use c_s to indicate the challenge. $H()$ is a cryptographic one-way hash synthesized using AES ($E()$).

TagFolio: The hash locks $L()$, are the same as before, using $H()$ to produce a *metaID* that can be unlocked using the one-time password that cannot be easily inverted and can be transmitted over an insecure medium.

TagDirective: TagDirective has the same initial requirements as TagCheck and TagFolio.

Kill Resistance: For Exponential Rampup, the delay is denoted with difficulty d and d_0 . TagDirective and Exponential Rampup will share the timer τ for limiting token usability and resetting the kill difficulty. The challenge c_z is the solution to the puzzle given to a kill request.

8.2.2 Tag Operations within TinFoil

Now that we have integrated all the individual elements into the TinFoil complete package, let us examine the capabilities of TinFoil as a whole. TinFoil allows us to accomplish the major tag operations listed below, all of which address the privacy and security concerns we have been discussing. The specific commands and tag responses necessary to complete the following operations are given in Table 8.2.

DISGUISE: change the public identity (PID) of the tag

REVEAL: reveal the private/real identity (RID) of the tag

AUTHENTICATE: prove that the real identity is genuine

HIDE: temporarily silence the tag

CHANGE: change authentication access control secrets such as hash locks

KILL: place the tag in a permanently silenced or deactivated mode

System	Symbol	Memory Type	Size (bits)	Size (transistors)
TagCheck	s_s	OTP	128	896
TagCheck	$H()$	AES hash	128	16,300
TagCheck	c_s	RAM	128	896
TagFolio	$L()$	FlashRAM	384	2,688
TagDirective	c_d	RAM	128	896
TagDirective	τ	RAM	16	112
TagDirective	$E()$	AES	N/A	0
Neighborhood Watch	N/A	N/A	N/A	0
Exponential Rampup	$H()$	AES hash	128	(Same as above)
Exponential Rampup	c_z	RAM	128	896
Exponential Rampup	d	RAM	7	49
Exponential Rampup	d_0	RAM	7	49
Total				22,782

s_s : static secret

$H()$: cryptographic hash function

c_s : challenge used by TagCheck

$L()$: TagFolio hash locks

c_d : challenge used by TagDirective

τ : time delay constant for Exponential Rampup

$E()$: cryptographic block encryption function

c_z : challenge used by Exponential Rampup

d : Exponential Rampup current puzzle difficulty

d_0 : Exponential Rampup initial puzzle difficulty

Table 8.1: TinFoil tag capabilities and size

Given the possible operations we have just described, there are three distinct states that the TinFoil JanusTag can be in. These are:

Dead: Tag does not respond to any input. Chip deactivates the power supply and transmission module. All writable areas are set to 0. If RID is writable (in certain implementations), it is also set to 0.

Hidden: Tag does not participate in anti-collision and will only respond when directly addressed with RID and given any $L()$.

Public: Tag participates in anti-collision and will respond with PID. If the given command contains a $L()$, the tag will perform the appropriate operation.

Command	Argument	Tag Response
normal query	none	return PID
validate $L_1()$	will vary	change RID to argument
validate $L_2()$	none	return RID
validate $L_2()$	AUTHENTICATE	authenticate using TagCheck
validate $L_3()$	SECRET	set value of s_s (once)
validate $L_3()$	HASH LOCK	change hash locks to arguments 1,2,3
validate $L_3()$	SETPUZZLE	set d_0 and d to arguments 1 and 2
validate $L_3()$	SILENCE	silence tag
validate $L_3()$	UNSILENCE	unsilence tag
validate $L_3()$	KILL	start TagDirective protocol for the kill operation
WARNING	none	activate kill resistance mechanism Exponential Rampup

Table 8.2: Tag responses to issued commands inside the TinFoil environment

8.3 Threat Analysis

In this section, I analyze the ability of TinFoil to resist the RFID attacks described in Section 3.4.

8.3.1 Denial of Service Attacks

8.3.1.1 Draining a Tag's Power Source

The TinFoil JanusTag is assumed to be a passive tag. As a result, such an attack is not possible.

8.3.1.2 Rogue Blockers

If a TinFoil JanusTag makes use of the tree-walking anti-collision mechanism, it is equally susceptible to a blocker as a regular RFID tag. However, the ability of the JanusTag to change its identity via a mutable PID does open up interesting possibilities for use in the presence of a selective blocker tag which only protects certain tags. When present, the blocker will make certain tags unreadable if they are within a given tag-space. If a user chooses to place the PID in that tag space, the blocker prevents it from being read. If the PID is not in the protected space, it is visible.

8.3.1.3 Mass Killing

TinFoil's major advantage is its resistance to mass kills. An authorized reader can kill a tag during the window when its time-limited challenge c_f is valid for each token,

but after time limit τ has passed, the token is no longer valid when the Deadline Timer resets the tag's memory of the challenge.

Even assuming that an attacker was able to acquire the secret(s) to a large group of tags, damage is still limited. Each time the attacker kills a tag, the other tags in the area will double the difficulty of their puzzle. As a result, the attacker will be exponentially slowed down during the time period set for τ . Furthermore, while the attacker is killing tags, any nearby readers will notice that the kill authorizations are not signed by a TagDirective server and will instruct other tags in the area to increase their kill difficulty and quench the attack.

Needless to say, if an attacker can isolate a tag from all other tags and readers to keep the kill signal from being detected, they will succeed in killing the tag. However, the need for such tag isolation makes killing large numbers of tags an arduous process.

8.3.1.4 RF Jamming

Strong RF jamming will continue to cause problems with a TinFoil system, same as with any other. If a reader uses a separate wireless channel for its connections to the various servers, RF jamming will also prevent operation over those channels. Conveniently, TagCheck's decoupled authentication allows it to complete its server operations once the jamming has subsided even if the tag is no longer present.

8.3.2 Man-in-the-Middle (MITM) Attacks

Assuming that the connection between a TagFolio and the Internet is secured by proper use of TLS/SSL or an equivalent mechanism, monitoring the link for information other than when data transmission occurs is difficult.

8.3.2.1 Passive Monitoring and Skimming

An attacker can always obtain the JanusTag's PID unless the tag is placed in the **Hidden** mode.

8.3.2.2 Replay Attacks

TinFoil's use of timers, randomly generated challenges and other similar protections in its protocols render replay attacks obsolete. In the case of TagCheck, an attacker can fool the server into thinking that a reader wished to check a particular tag, but they can't do much else. TagFolio transmits hash locks exclusively within a secured environment, reducing the chances that such transactions can be monitored. The nature of a hash lock itself makes replay attacks difficult as the secret and the lock are different. TagDirective uses tag-generated challenges to prove the tag's liveness and to decrypt the access control token that passed back to it.

8.3.2.3 MultimediuM Attacks

As stated above, multimediuM attacks are not possible in the case of TagCheck, TagFolio, or TagDirective because the reader-to-server link is separately authenticated and encrypted.

However, we must still consider the TorONS link, which uses a separate mechanism, The Onion Router, to achieve anonymization. The Onion Router provides an encrypted link between the application and the Tor service, but not to the final endpoint³. This means that an attacker, who has sufficient resources to analyze traffic and track packets going between the application and the various Tor servers, might be able to see the ONS look-ups on the last link in the chain. However, this can be made significantly harder by changing the Tor route frequently or by having multiple Tor routes open simultaneously. An attacker who is able to watch multiple geographically-separated Tor nodes has resources far greater in value than the information we are trying to protect, creating a highly unlikely scenario.

Another possible multimediuM attack involves trying to detect the presence of an RFID tag by looking for its powered coil. If a JanusTag is designed to use the gadolinium nitrate self-destruction mechanism from [GR02], it can physically sever the antenna coil's connection and consequently make such an attack less relevant. Even if the tag is unable to disconnect the coil, its detection can be made more difficult if the transmitter is not powered up.

Two other previously mentioned attacks, Differential Power Analysis (DPA) [KJJ99] and Differential Electro Magnetic Analysis (DEMA) [CLRP05], are valid techniques for discerning tag secrets. In the **Dead** mode, a tag does not respond or power up, so in this mode both these attacks are irrelevant. To protect the tag while it is alive, for a small increase in cost, we can make use of a power regulator. However, since DPA and DEMA techniques are still beyond the current capability of a standard attacker, we will not consider them to be a significant threat for any but the most security-sensitive applications. For sufficiently sensitive applications, the addition of a power regulator should be considered.

8.3.3 Social Attacks

Functional Requirements 11 and 12 are designed to make social attacks significantly harder. The reader and TagFolio are not allowed access to each other's secrets during their transactions. Instead an intermediary completes ownership transaction by applying new hash locks to prevent the previous owner from continued access. Such prevention requires the use of secure hash functions or block ciphers.

TagDirective access control uses an equivalent token in deference to transferring a static secret. This lack of an exposed secret protects the devices and their owners

³which would require rewriting various protocols that did not intend to use encryption including DNS

from losing the secrets or having them stolen. Not transferring tag secrets allows security to be preserved even when these static secrets are reused, intentionally or not.

8.4 Illustration of TinFoil's Capabilities: A Medical Case Study

To fully appreciate the power of TinFoil, I will illustrate its capabilities with an example. Let us return to the medical case study introduced in Section 3.3. Our friend, Robert Paulson, is back at the pharmacy and has just obtained a bottle of Bertox to treat his medical condition. This time, however, the RFID tag embedded in the pill bottle is a JanusTag.

Yet again, Bob has to decide whether to keep the tag alive while transporting the bottle home. Last time, when the Bertox bottle contained a regular RFID tag, Bob chose to kill the tag out of concern for his privacy. The JanusTag in his new pill bottle, however, changes his decision. Since he used TagFolio to purchase Bertox, the ID of the JanusTag was encrypted upon the transaction, making it very difficult for anyone except for authorized readers to decipher. Using his TagFolio, Bob was even able to verify that the tag was reprogrammed before leaving the store.

When he arrives at home, Bob first wants to make sure his drug is genuine. Since Foley Pharmaceuticals is complying with the current legislation on drug pedigrees, the information about Bob's particular bottle of Bertox is stored in their database. Using his TagCheck-enabled reader to first authenticate the tag in his bottle⁴, Bob is able to access this information through a web interface to confirm that the serial number does in fact match the bottle in his hand and that the pedigree is good. Depending on the vendor, Bob may even be presented with the tracking and manufacturing information from beginning to finish. To allay his fears that someone watching his Internet connection may find out he is taking Bertox, Bob uses TorONS to access all the online information.

Reassured that his drug is genuine, Bob employs his home automation system to help him remember to take his medication. He also sets up his TagFolio to allow his wife Alice access to his data, and, in particular, to the keys for accessing other data. If he wanted to, he could instead give a special key to a third party such as a lawyer to only be distributed upon his death. However, Bob trusts Alice and thus grants her access to his data once she installs the proper keys on her TagFolio. Using these keys, Alice can see what medications Bob is taking. Incidentally, these are the same keys used by the previously mentioned home automation system to help Bob comply with the medication regime.

⁴This authentication step solves the problem of someone copying labels with known good serial numbers and putting them on counterfeit bottles of medicine.

While driving to work one day, Bob has a heart attack. The paramedics who arrive on-scene wish to view Bob's medical records to find out what drugs he may be taking and other relevant information. They find Bob's TagFolio, plug it into their Internet-enabled document viewing device and access TagDirective. TagDirective uses the federated access control model to authenticate the paramedics to Bob's medical record storage area and release the information the paramedics are allowed to have: medicines, recent medical history, allergies, emergency contact information, etc.

Despite the availability of information to the paramedics and consequently the best possible medical care, Bob still dies from the heart attack. Just like last time, Alice believes that Bertox led to Bob's death and sues Foley Pharmaceuticals. However, since all the information about Bertox's manufacturing has been properly indexed using JanusTags, Foley Pharmaceuticals is able to track down information about Bob's specific pill bottle admitted as evidence and consequently defend itself by showing that proper FDA procedures had been followed. The electronic trail also allows Foley Pharmaceuticals to minimize the exposure of trade secrets during the trial. The good pedigree protects the company in court, or, at a minimum, lowers the out-of-court settlement.

Once the trial is over, the last remaining task is to dispose of the old bottle of Bertox. Since Alice has the key that the JanusTag in the bottle was encrypted with, she can use her TagFolio to decrypt the tag's ID and set it to broadcast the real ID. This will allow the processing plant to sort the bottle correctly for recycling.

8.5 Summary

In this body of work, I have developed a complete secure EPC architecture called TinFoil to address the privacy and security concerns inherent to a unique-identifier environment. TinFoil combines a number of components, each of which protects a different aspect of the RFID interaction. On the business level, I proposed TagCheck, a counterfeit-detection system to minimize the possibility of identifier theft, and TagFolio, an ownership-transfer system to reduce the possibility of data leakage. I designed kill-resistance solutions such as the Neighborhood Watch community and Exponential Rampup to protect tags from a mass kill attack. On a more personal level, JanusTag allows ownership to safely recode and dynamically change tag presence. TorONS anonymization dissociates identifier queries from entities while still allowing resolution to occur. TagDirective is effective in both business and personal environments to manage secrets and provide time-limited access control.

To analyze the potential weaknesses of TinFoil, I explored its ability to withstand some of the common RFID exploits. I found that TinFoil significantly reduces secret leakage while adding only minimal additional cost to the RFID system.

To illustrate TinFoil's full capabilities, I analyzed the functionality of the system using scenarios drawn from typical practical applications including the military and

medical fields. I conclude that the TinFoil package is well-suited to forming a core platform for building a secure and reliable RFID-enhanced environment at a relatively low cost.

Appendix A

Related Work

This chapter contains information about other related work that do not fit within the main focus of the thesis, but are still important for the understanding of the domain.

A.1 Yellow Pages (YP)

Yellow Pages (YP) [nis], now known as Network Information Service (NIS) is a general directory look-up service, provided by Sun as another alternative to DNS. It is capable of returning standard address information and could return general table look-up information.

YP did construct some interesting concepts that were slightly different than DNS. As with most of these systems, YP defined a set of key-value pairings, but these pairings could be grouped together into maps. A domain within YP is a category of maps. These three concepts of key-value, map, and domain allowed YP to model most simple database information.

YP also had the concept of referral through a mechanism called YPbind. YP uses YP binder servers to bootstrap the process of finding the correct YP servers to query for further information. Since YPbind was considered an optional part and is not very scalable, it is clear that the system was designed mostly for intranets.

A.2 Lightweight Directory Access Protocol (LDAP)

Lightweight Directory Access Protocol (LDAP) is another query system that was supported by Microsoft, IBM, Lotus, and Netscape to try to centralize a directory [lda].

LDAP has a general purpose query language that includes method for specifying filters and the formatting of the output. A query makes use of a key, which causes the system to return a number of name-value pairings. There is also a permission system to restrict access to only authorized users.

LDAP was designed at the University of Michigan and was based upon an enterprise directory system called X.500. X.500 itself was not used as it was too heavyweight a protocol and the server itself was a bit much for the PCs available at the time.

At this time, LDAP is used mostly by modern email clients to associate information with and to look-up email addresses, but not much else. Part of the problem is that LDAP was designed for a centralized service, which has issues on massive-level scaling.

A.3 Universal Description, Discovery and Integration (UDDI)

A more web-focused discovery protocol is Oasis's Universal Description, Discovery and Integration (UDDI) protocol initially developed in 2000. To quote their website [Oas]:

“UDDI creates a standard inter-operable platform that enables companies and applications to quickly, easily, and dynamically find and use Web services over the Internet. UDDI also allows operational registries to be maintained for different purposes in different contexts.”

This system is a XML based query and response search system, similar in mind to SOAP. However, it also makes use of the more general publish-subscribe system to distribute constantly updating information in an efficient manner. This is the part of interest for any modern design of a discovery system, as the amount of data that needs to be kept updated is significant.

A concept from UDDI that our system makes use of is internal and external views. Users can see either an internal or external view depending upon their authorization. This provides a measure of security granularity that is useful for grouping data and its interpretation together. The way that the data is managed among the different views is also quite interesting.

In the latest version 3, the registry services have been broken down into three types [Oas04]:

Corporate/Private: Highly valuable information in an internal registry. Firewalled to prevent access from the public network. Administrative features and the data itself are carefully restricted. No sharing with other registries.

Affiliated: Controlled environment, but with some limited access by authorized parties outside of the local network. Administration may be delegated to trusted parties. Data may be shared with other registries to a limited degree.

Public: Free access to data. Administrative functions restricted. Data may be shared with other registries, but the content may also be moderated.

A.4 Secure Sockets Layer (SSL)

Secure Sockets Layer *a.k.a.* Transport Security Layer is a cryptographic protocol and abstraction designed to protect the interface between the browser and the web-server [TLS]. SSL makes use of both symmetric and public-key cryptography to authenticate then encrypt communications. Unfortunately, this protocol relies heavily on having a general purpose fast CPU and a significant amount of memory. These things are not applicable to the tag and reader interface because making a tag capable of communicating via SSL would put such high requirements on computation and power as to make it useless on a tag.

However, SSL is a very good choice for protecting the link between the reader and the various servers on the Internet that it must communicate with. Unfortunately, SSL does not give a simple solution to the Public Key Infrastructure problem that is often posed, but it does give some tools to make addressing such a problem reasonable. An SSL system depends upon a public list of Signature Authorities that authorize the various certificates that are used for individual authorization.

A.5 Zero-Knowledge Proofs (ZKPs)

A zero-knowledge proof system is an interesting cryptographic construct that allows an entity to prove that they possess a particular piece of information without sharing that specific information.

An simple explanatory example, given in [QG90], is that of Ali Baba's cave shown in Figure A-1.

The premise is that Alice knows the secret password that can open the secret passageway that allows one to pass between R and S. She wants to prove this to Bob, but telling him the password would mean that he can now open the passageway, an undesirable level of trust to Alice. Alice proves that she knows it by going into the cave and picking a random path to either R or S. Bob then enters the cave only to Q. Bob then tells her to come out of the right passageway (R) or the left (S). Since Alice can open the passageway, she can always do this. Someone who did not know the secret password would only be able to succeed 50% of the time since they would be stuck with whichever path they chose. After doing n rounds of this game, Bob should be able to arrive at a reasonable level of certainty $(1 - 2^{-n})$ that Alice knows the secret.

A zero-knowledge proof would be extremely helpful to the authentication problem presented here. Unfortunately, many zero-knowledge proofs such as Fiat-Shamir [FS87] are memory and computationally intensive. This makes them inapplicable to the computationally unstable and restrictive environment of an RFID tag. However, the important piece we need in order to make a workable system is a cryptograph-

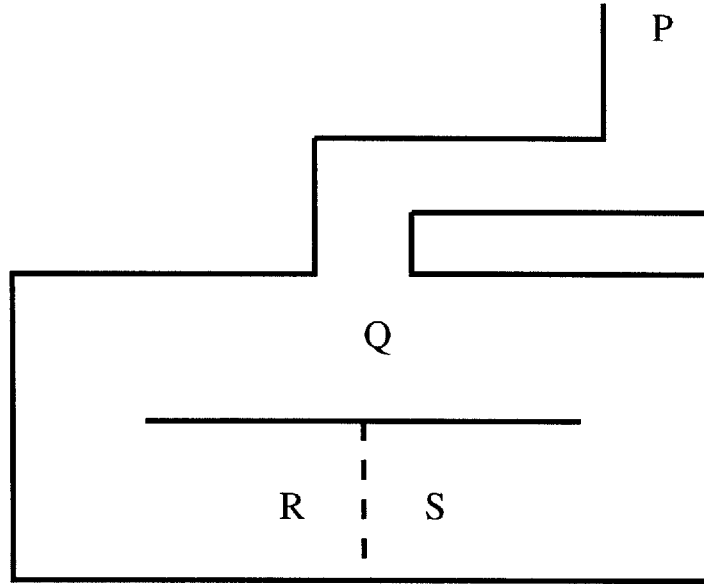


Figure A-1: Ali Baba's strange cave

ically secure one-way¹ hash function “H()” that can be implemented easily on the RFID tag.

A.6 Leap of Faith

Instead of focusing on the cryptography, they propose having the focus be on protecting interception of a message at its entry into the Internet and as it is about to get to its final destination. The justification for this simplification is that Man-in-the-Middle (MITM) attacks are most often applied and most useful at these points. The likelihood that an attacker has hacked into a major Internet router to look for a particular message is very unlikely². This work also divides trust augmentation into two categories:

Public Key Infrastructure (PKI): Trust is distributed from a small number of trusted certificate authorities who can sign key-pairings.

Leap of Faith: Each participant keeps track of its own keys without use of a centralized server of any sort.

[BS05] proposes that a middle ground exists that is very reminiscent of the PGP “Web of Trust.” This method is called Random Buddy Verification. Random Buddy

¹Such a hash function is also known as a trap-door function, making it very reminiscent of the Ali Baba analogy.

²with much larger ramifications

Verification starts by having the computer trying to make a secure channel to another computer meet other computers. This works particularly well for mobile devices, which should interact with a large number of other computers on a “first-hop” basis. It exchanges a key pair with each of them and saves it for later, remembering them as buddies.

When the original computer starts to make a connection to a remote destination by requesting a public key, it also contacts a random sampling of its buddies to do the same and return those keys. Assuming that at least one of these buddies is not on the same path as a MITM attack, it should get the correct key. When a disparity emerges, the user can be warned of this situation and take corrective measures. Unfortunately, this only protects from MITM attacks on the first-hop. In order to protect the rest of the chain, the remote server needs to request the public key of the originating server and repeat the previous process.

Needless to say, this design is not for paranoid applications such as financial or espionage, however it may be sufficient for bootstrapping less critical applications like reader-to-reader authentication.

A.7 Intuitive RFID User Interface

Regardless of any given solution to a particular privacy and security system that is proposed, a problem that always arises is that of user interface. How will a typical user interface with the security system? The more intuitive the system is, the more likely that an average person will be able to intelligently apply it and protect their valuable data.

One development that may assist with this is described in [RSA06]. Riekkilä *et al.* designed a system where by a person with a mobile device touches the device against a tag with an explanatory icon printed on it to enable a given service. With very little training, people were able to figure out how to use it for tasks such as printing, making a phone call or looking up a map.

Appendix B

Code Distribution

B.1 TorLib

TorLib is a Java reimplementation of standard Java network functions that have been wrapped to use the Tor anonymization network. TorLib can be found as part of the TinFoil project at <http://web.mit.edu/foley/TinFoil/>.

I will now present an example of using the TorLib code to look up DNS names or get files specified with URLs in Section B.2.1. This program was used to gather statistics on the time it took to look up a large number of hosts. Also included is a perl program used for testing the Tor `tor-resolve` client in Section B.2.2. At the end of this appendix is the rest of the TorLib code that interfaces with dnsjava [Wel].

B.2 Demonstrations and Profiling Clients

B.2.1 TorONS.java

```
import java.io.*;
import java.util.*;
import java.net.InetAddress;
import java.net.Socket;
5 import TorLib.*;
import org.xbill.DNS.DClass;
import org.xbill.DNS.Type;
import org.xbill.DNS.Record;
import org.xbill.DNS.Name;
10
public class TorONS {
    public static void main(String[] args) throws IOException {
        {
```

```

String targetHostname = "tor.eff.org";
String targetDir = "index.html";
15 int targetPort = 80;

if(args.length < 1 || args.length > 0 && args[0].equals(\
    "-h")) {
    System.out.println("TorONS_\interface_for_testing/\
        using_Tor_from_Java_\n"+
20         "for_EPC_Applications_\n"+
        "By_Joe_Foley<foley@mit.edu>\n"+
        "Usage:_java_Tinfoil.TorONS_<cmd>_<\
            args>\n"+
        "<cmd>_can_be:_-h_for_help_\n"+
        "_____r_for_resolve_\n"+
        "_____f_<file>_for_\
            resolve_hostnames_in_<file>\n"+
25         "_____w_for_webget_\n"+
        "For_r,_the_arg_is:\n"+
        "___<hostname1>_<hostname2>_..._\
            Hostnames_to_DNS_resolve_\n"+
        "For_w,_the_args_are:\n"+
        "___<host>_<path>_<optional_port>\n\
            "+
30         "_for_example,_http://tor.eff.org\
            :80/index.html_would_be_\n"+
        "___tor.eff.org_index.html_80_\n"+
        "_Since_this_is_a_demo,_the_default\
            _is_the_tor_website._\n");

    System.exit(2);
}
35

String req = args[0];
if(req.equals("-r")) {
    for(int i = 0; i < args.length; i++) {
        System.out.println(TorLib.TorResolve(args[i]));
40    }
}
else if(req.equals("-f")) {
    File hostfile = new File(args[1]);
    System.out.println("Reading_hostnames_from"+args[1]);
45    if(! hostfile.exists()) {
        abort("Error:_no_such_file:_\""+args[2]+"\"");
    }
}

```

```

    }
    if (! hostfile.isFile()) {
        abort("Error: \"+args[2]+\" \"is not a file. \\
            Perhaps it is a directory?");
50    }
    if (! hostfile.canRead()) {
        abort("Error: file unreadable: \"+args[2]+\"");
    }

55    int type = Type.A;

    // try opening the file
    BufferedReader in = new BufferedReader(new FileReader(\
        hostfile));

60    // We're all setup, let time this bad boy
    Date startd = new Date();
    String hostname;
    Vector hostnames = new Vector();

65    while((hostname = in.readLine()) != null) {
        System.out.println("host: \" + hostname);
        hostnames.add(hostname);
    }

70    TorLookup l = new TorLookup(hostnames, Type.A, DClass.\
        IN);
    Vector answers = l.runs();
    Date endd = new Date();
    TorLookup.printAnswers(answers, l);

75    long startt = startd.getTime();
    long endt = endd.getTime();
    long difference = endt - startt;
    System.out.println("Total: \" + difference + \" \
        milliseconds");
    }
80    else if(req.equals("-w")) {
        if (args.length >= 4)
            targetPort = new Integer(args[2]).intValue();
        if (args.length >= 3)
            targetDir = args[2];

```

```

85     if (args.length >= 2)
        targetHostname = args[1];
    if (args.length >= 1)
        req = args[0];

90    try {
        Socket s = TorLib.TorSocket(targetHostname, \
            targetPort);
        DataInputStream is = new DataInputStream(s.\
            getInputStream());
        PrintStream out=new java.io.PrintStream(s.\
            getOutputStream());

95        //Construct an HTTP request
        String getreq = "GET_/_"+targetDir+"_HTTP/1.0\r\n"+
            "Host:_"+targetHostname+": "+targetPort+"\r\n"+
            "Accept:_*/*\r\n"+
            "Connection:_Keep-Alive\r\n"+
100        "Pragma:_no-cache\r\n"+
            "\r\n";
        out.print(getreq);
        System.err.println("Sending:\n"+getreq);
        out.flush();

105        // this is from Java Examples In a Nutshell
        final InputStreamReader from_server = new \
            InputStreamReader(is);
        char[] buffer=new char[1024];
        int chars_read;

110        // read until stream closes
        while((chars_read = from_server.read(buffer)) != -1)\
        {
            // loop through array of chars
            // change \n to local platform terminator
            // this is a nieve implementation
115            for(int j=0; j<chars_read; j++) {
                if(buffer[j]=='\n') System.out.println();
                else System.out.print(buffer[j]);
            }
            System.out.flush();
120        }
    }

```

```

        s.close();
    }
    catch (Exception e) {e.printStackTrace(); }
125 }
}

/* From Java Examples in a Nutshell */
private static void abort(String msg) throws IOException {
130     throw new IOException("TorONS:_" + msg);
}
}

```

B.2.2 test-tor-resolve.pl

```

#!/usr/bin/perl -w
use Time::HiRes qw (clock_gettime clock_getres \
    CLOCK_REALTIME gettimeofday tv_interval) ;
3
my $torresolve = "/usr/local/bin/tor-resolve";
my $infile = shift || "hostnames.txt";
# this file contains our test hostnames
my $start = [gettimeofday];
8
#print "Start: $start (s: $starts us: $startu)\n";
open(HOSTFILE, $infile) or die "Unable to open $infile for \
    reading: _$_!\n";
while(<HOSTFILE>) {
    chomp;
13    my $hostname = $_;
    system("$torresolve _$hostname");
}
my $end = [gettimeofday];
#print "End: $end (s: $ends us: $endu)\n";
18 my $time = tv_interval $start, $end;
print "Total: _$time_seconds\n";

```

B.3 TorLib

This code is based heavily upon work by Brian Wellington's `dnsjava`. Some of the code is taken directly from `dnsjava`, with permission of the author.

B.3.1 TorLib.java

This java code takes care of miscellaneous useful functions.

```

1 package TorLib;
  /**
   * Tinfoil is an RFID Privacy and Security Enhancement library.
   *
   * Copyright (c) 2005 Joe Foley, MIT AutoID Labs
6
   Permission is hereby granted, free of charge, to any person \
     obtaining a
     copy of this software and associated documentation files (the \
       "Software"),
     to deal in the Software without restriction, including \
       without limitation
     the rights to use, copy, modify, merge, publish, distribute, \
       sublicense,
11  and/or sell copies of the Software, and to permit persons to \
       whom the
     Software is furnished to do so, subject to the following \
       conditions:

     The above copyright notice and this permission notice shall \
       be included
     in all copies or substantial portions of the Software.
16
     THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY \
       KIND, EXPRESS
     OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF \
       MERCHANTABILITY,
     FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO \
       EVENT SHALL THE
     AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES \
       OR OTHER
21  LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR \
       OTHERWISE, ARISING FROM,
     OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER \
       DEALINGS
     IN THE SOFTWARE.

   */
  // $HeadURL: file:///afs/athena.mit.edu/user/f/o/foley/Classes/\
    SVNRepository/TinFoilSVN/TinFoil/TorLib/TorLib.java $
26 // $Id: TorLib.java 21 2005-12-06 09:33:07Z foley $

import java.io.DataInputStream;
```

```

import java.io.DataOutputStream;
import java.io.IOException;
31 import java.io.InputStreamReader;
import java.io.PrintStream;
import java.net.InetAddress;
import java.net.Socket;

36 /**
 * The Onion Router(Tor) Java Library routines<br />
 * It is part of the TinFoil RFID Privacy/Security system.<br />
 * These methods allow us to setup an anonymized TCP socket \
   through
 * the Tor network and do safe anonymized DNS lookups.<br />
41 * This code was written with the help of Roger Dingledine and \
   Nick Mathewson.<br />
 * The code is open-source under the MIT X11 license.<br />
 * Documentation on the API can be found at <a href="http://web.\
   mit.edu/foley/www/TinFoil/Docs/">http://web.mit.edu/foley/www\
   /TinFoil/Docs/</a>
 *
 * <ul>
46 * <li><a href = "http://tor.eff.org"> http://tor.eff.org</a>
 * <li><a href = "http://tor.eff.org/cvs/control/doc/howto.txt">\
   http://tor.eff.org/cvs/control/doc/howto.txt</a>
 * <li><a href = "http://www.mokabyte.it/2000/06/firewallutil.htm\
   ">http://www.mokabyte.it/2000/06/firewallutil.htm</a>
 * </ul>
 *
51 * @author Joe Foley<foley at MIT dot EDU>, MIT AutoID Labs
 * @version 1.0
 * <p>
 */
public class TorLib {
56     /**
     * Default TOR Proxy port.
     */
     static int proxyPort = 9050;
     /**
61     * Default TOR Proxy hostaddr.
     */
     static String proxyAddr = "localhost";
     /**
     * Constant tells SOCKS4/4a to connect. Use it in the <i>\
       req</i> parameter.
66     */

```

```

final static byte TOR_CONNECT = (byte) 0x01;
/**
 * Constant tells TOR to do a DNS resolve. Use it in the\
 * <i>req</i> parameter.
 */
71 final static byte TOR_RESOLVE = (byte) 0xF0;
/**
 * Constant indicates what SOCKS version are talking
 * Either SOCKS4 or SOCKS4a
 */
76 final static byte SOCKS_VERSION = (byte) 0x04;
/**
 * SOCKS uses Nulls as field delimiters
 */
final static byte SOCKS_DELIM = (byte) 0x00;
81 /**
 * Setting the IP field to 0.0.0.1 causes SOCKS4a to
 * be enabled.
 */
final static int SOCKS4A_FAKEIP = (int) 0x01;
86
/**
 * This method allows you to demo/access the resolver\
 * socket generation from
 * the command line. Run with "-h" to get the help menu.
 * @param args Command line arguments for test main \
 * method.
91 */
public static void main(String[] args) {
    String req = "-r";
    String targetHostname = "tor.eff.org";
    String targetDir = "index.html";
96    int targetPort = 80;

    if (args.length > 0 && args[0].equals("-h")) {
        System.out.println("Tinfoil/TorLib_\n\
            interface_for_using_Tor_from_Java\n"+
            "By_Joe_Foley<foley@mit.\n\
            edu>\n"+
            "Usage: _java_Tinfoil.\n\
            TorLib_<cmd>_<args>\n"\n\
            +
            "<cmd>_can_be: _-h_for_\n\
            help\n"+

```

```

106         " _ _ _ _ _ _ _ _ _ _ -r _for _\
            resolve\n"+
        " _ _ _ _ _ _ _ _ _ _ -w _for _\
            wget\n"+
        "For _-r , _the _arg _is :\n"+
        " _ _ <hostname> _Hostname _to\
            _DNS _resolve\n"+
        "For _-w, _the _args _are :\n"\
        +
        " _ _ _ <host> _<path> _<\
            optional _port>\n"+
        " _for _example , _http://tor\
            .eff.org:80/index.html\
            _would _be\n"+
111         " _ _ _tor.eff.org _index.\
            html _80\n"+
        " _Since _this _is _a _demo, _\
            the _default _is _the _tor\
            _website.\n");
        System.exit(2);
    }

116     if (args.length >= 4)
        targetPort = new Integer(args[2]).\
            intValue();
    if (args.length >= 3)
        targetDir = args[2];
    if (args.length >= 2)
121         targetHostname = args[1];
    if (args.length >= 1)
        req = args[0];

    if(req.equals("-r")) {
126         System.out.println(TorResolve(\
            targetHostname));
    }
    else if(req.equals("-w")) {
        try {
            Socket s = TorSocket(\
                targetHostname , targetPort);
131         DataInputStream is = new \
            DataInputStream(s.\
                getInputStream());
            PrintStream out=new java.io.\
                PrintStream(s.getOutputStream\

```

```

    ());

    // Construct an HTTP request
    String getreq = "GET_/_"+
        targetDir+"_HTTP/1.0\r\n"+
136    "Host:_"+targetHostname+":_"+
        targetPort+"\r\n"+
        "Accept:_*/_*\r\n"+
        "Connection:_Keep-Alive\r\n"+
        "Pragma:_no-cache\r\n"+
        "\r\n";
141    out.print(getreq);
        System.err.println("Sending:\n"+
            getreq);
    out.flush();

    // this is from Java Examples In a \
    Nutshell
146    final InputStreamReader \
        from_server = new \
        InputStreamReader(is);
    char[] buffer=new char[1024];
    int chars_read;

    // read until stream closes
151    while((chars_read = from_server.\
        read(buffer)) != -1) {
        // loop through array of \
        chars
        // change \n to local \
        platform terminator
        // this is a nieve \
        implementation
        for(int j=0; j<chars_read\
            ; j++) {
156            if(buffer[j]=='\n\
                ') System.out.\
                println();
            else System.out.\
                print(buffer[j\
                    ]);
        }
        System.out.flush();
    }
161    s.close();

```

```

        }
        catch (Exception e) {c.printStackTrace();\
        }
    }
}

166

/**
 * This method Creates a socket, then sends the initial \
 * SOCKS request info
 * It stops before reading so that other methods may
171 * differently interpret the results. It returns the \
 * open socket.
 *
 * @param targetHostname The hostname of the destination \
 * host.
 * @param targetPort The port to connect to
 * @param req SOCKS/TOR request code
176 * @return An open Socket that has been sent the SOCK4a \
 * init codes.
 * @throws IOException from any Socket problems
 */
static Socket TorSocketPre(String targetHostname, int \
    targetPort, byte req)
throws IOException {
181     Socket s = new Socket(proxyAddr, proxyPort);
    return(TorSocketPre(s, targetHostname, targetPort\
        , req));
}

/**
186 * This Initialization routine is for if someone else \
 * opened the socket for us.
 * @param s
 * @param targetHostname
 * @param targetPort
 * @param req
191 * @return
 * @throws IOException
 */
static Socket TorSocketPre(Socket s, String \
    targetHostname, int targetPort, byte req)
throws IOException {
196

```

```

System.out.println("Opening connection to "+
    targetHostname+": "+targetPort+
        " via proxy "+proxyAddr+": "+
            proxyPort+" of type "+req);
DataOutputStream os = new DataOutputStream(s.\
    getOutputStream());
os.writeByte(SOCKS.VERSION);
os.writeByte(req);
// 2 bytes
os.writeShort(targetPort);
// 4 bytes, high byte first
os.writeInt(SOCKS4A_FAKEIP);
os.writeByte(SOCKS.DELIM);
os.writeBytes(targetHostname);
os.writeByte(SOCKS.DELIM);
return(s);
}

/**
 * This method creates a socket to the target host and \
 * port using TorSocketPre, then reads
 * the SOCKS information.
 * @param targetHostname Hostname of destination host.
 * @param targetPort Port on remote destination host.
 * @return Fully initialized TCP Socket that tunnels to \
 * the target Host/Port via the Tor Proxy host/port.
 * @throws IOException when Socket and Read/Write \
 * exceptions occur.
 */
static Socket TorSocket(String targetHostname, int \
    targetPort)
throws IOException {
    Socket s = TorSocketPre(targetHostname, targetPort \
        ,TOR.CONNECT);
    return(TorSocketPost(s));
}

static Socket TorSocket(Socket s, String targetHostname, \
    int targetPort)
throws IOException {
    TorSocketPre(s, targetHostname, targetPort, \
        TOR.CONNECT);
    return(TorSocketPost(s));
}

```

```

static Socket TorSocketPost(Socket s)
throws IOException {
    DataInputStream is = new DataInputStream(s.\
        getInputStream());
236    // only the status is useful on a TOR CONNECT
    byte version = is.readByte();
    byte status = is.readByte();
    if(status != (byte)90) {
        //failed for some reason, return useful \
            exception
241        throw(new IOException(ParseSOCKSStatus(\
            status)));
    }
    //      System.out.println("status: "+\
        ParseSOCKSStatus(status));
    int port = is.readShort();
    int ipAddr = is.readInt();
246    return(s);
}

/**
251    * This method opens a TOR socket, and does an anonymous \
        DNS resolve through it.
    * Since Tor caches things, this is a very fast lookup if\
        we've already connected there
    * The resolve does a gethostbyname() on the exit node.
    * @param targetHostname String containing the hostname \
        to look up.
    * @return String representation of the IP address: "x.x.\
        x.x"
256    */
static String TorResolve(String targetHostname) {
    int targetPort = 0; // we dont need a port to \
        resolve

    try {
261        Socket s = TorSocketPre(targetHostname,\
            targetPort,TOR_RESOLVE);
        DataInputStream is = new DataInputStream(\
            s.getInputStream());

        byte version = is.readByte();
        byte status = is.readByte();

```

```

266         if(status != (byte)90) {
            //failed for some reason, return \
            useful exception
            throw(new IOException(\
                ParseSOCKSStatus(status)));
        }
        int port = is.readShort();
271     byte[] ipAddrBytes = new byte[4];
        is.read(ipAddrBytes);
        InetAddress ia = InetAddress.getByAddress\
            (ipAddrBytes);
        //System.out.println("Resolved into:"+ia)\
        ;
        is.close();
276     String addr = ia.toString().substring(1);\
        // clip off the "/"
        return(addr);
    }
    catch(Exception e) {e.printStackTrace();}
    return(null);
281 }

```

```

/**
286 * This helper method allows us to decode the SOCKS4 \
    status codes into
    * Human readable input.<br />
    * Based upon info from http://archive.socks.permeo.com/\
    protocol/socks4.protocol
    * @param status Byte containing the status code.
    * @return String human-readable representation of the \
    error.
291 */
static String ParseSOCKSStatus(byte status) {
    // func to turn the status codes into useful \
    output
    // reference
    String retval;
296     switch(status) {
        case 90:
            retval = status+"_Request_granted.";
            break;
        case 91:

```

```

301         retval = status+"_Request_rejected/failed\
            _unknown_reason.";
        break;
    case 92:
        retval = status+"_Request_rejected:_SOCKS\
            _server_cannot_connect_to_identd_on\
            the_client.";
        break;
306    case 93:
        retval = status+"_Request_rejected:_the_\
            client_program_and_identd_report_\
            different_user-ids.";
        break;
    default:
        retval = status+"_Unknown_SOCKS_status_\
            code.";
311    }
    return(retval);
}
}

```

B.3.2 TorLookup.java

This java code performs the lookup query.

```

/**
 *
 */
package TorLib;

5
import java.io.IOException;
import java.io.InterruptedIOException;
import java.net.UnknownHostException;
import java.util.ArrayList;
10 import java.util.HashMap;
import java.util.Iterator;
import java.util.List;
import java.util.Map;

15 import org.xbill.DNS.CNAMERecord;
import org.xbill.DNS.Cache;
import org.xbill.DNS.Credibility;
import org.xbill.DNS.DClass;
import org.xbill.DNS.DNAMERecord;

```

```

20 import org.xbill.DNS.Lookup;
import org.xbill.DNS.Message;
import org.xbill.DNS.Mnemonic;
import org.xbill.DNS.Name;
import org.xbill.DNS.NameTooLongException;
25 import org.xbill.DNS.Options;
import org.xbill.DNS.RRset;
import org.xbill.DNS.Rcode;
import org.xbill.DNS.Record;
import org.xbill.DNS.Resolver;
30 import org.xbill.DNS.ResolverConfig;
import org.xbill.DNS.SetResponse;
import org.xbill.DNS.TextParseException;
import org.xbill.DNS.Type;

35 /**
 * @author foley
 *
 */
public class TorLookup {
40
    protected static TorResolver defaultResolver;
    protected static Name [] defaultSearchPath;
    protected static Map defaultCaches;

45
    protected Resolver resolver;
    protected Name [] searchPath;
    protected Cache cache;
    protected boolean temporary_cache;
    protected int credibility;
50
    protected Name name;
    protected int type;
    protected int dclass;
    protected boolean verbose;
    protected int iterations;
55
    protected boolean foundAlias;
    protected boolean done;
    protected boolean doneCurrent;
    protected List aliases;
    protected Record [] answers;
60
    protected int result;
    protected String error;
    protected boolean nxdomain;
    protected boolean badresponse;
    protected String badresponse_error;

```

```

65      protected boolean networkerror;
      protected boolean timedout;
      protected boolean nametoolong;
      protected boolean referral;

70      protected static final Name [] noAliases = new Name[0];

      /** The lookup was successful. */
      public static final int SUCCESSFUL = 0;

75      /**
       * The lookup failed due to a data or server error. \
       * Repeating the lookup
       * would not be helpful.
       */
      public static final int UNRECOVERABLE = 1;

80      /**
       * The lookup failed due to a network error. Repeating \
       * the lookup may be
       * helpful.
       */
85      public static final int TRY_AGAIN = 2;

      /** The host does not exist. */
      public static final int HOSTNOTFOUND = 3;

90      /** The host exists, but has no records associated with \
       * the queried type. */
      public static final int TYPENOTFOUND = 4;

      public static synchronized void
      refreshDefault() {

95          try {
              defaultResolver = new TorResolver();
          }
          catch (UnknownHostException e) {
100             throw new RuntimeException("Failed to \
              initialize resolver");
          }
          defaultSearchPath = ResolverConfig.\
              getCurrentConfig().searchPath();
          defaultCaches = new HashMap();
      }

```

```

105      static {
          refreshDefault();
      }

110      /**
       * Gets the Resolver that will be used as the default by \
       * future Lookups.
       * @return The default resolver.
       */
      public static synchronized Resolver
115      getDefaultResolver() {
          return defaultResolver;
      }

      /**
120      * Sets the default Resolver to be used as the default by \
       * future Lookups.
       * @param resolver The default resolver.
       */
      public static synchronized void
125      setDefaultResolver(TorResolver resolver) {
          defaultResolver = resolver;
      }

      /**
       * Gets the Cache that will be used as the default for \
       * the specified
130      * class by future Lookups.
       * @param dclass The class whose cache is being retrieved\
       *
       * @return The default cache for the specified class.
       */
      public static synchronized Cache
135      getDefaultCache(int dclass) {
          DClass.check(dclass);
          Cache c = (Cache) defaultCaches.get(Mnemonic.\
              toInteger(dclass));
          if (c == null) {
              c = new Cache(dclass);
140              defaultCaches.put(Mnemonic.toInteger(\
                  dclass), c);
          }
          return c;
      }

```

```

145      /**
        * Sets the Cache to be used as the default for the \
        * specified class by future
        * Lookups.
        * @param cache The default cache for the specified class\
        *
        * @param dclass The class whose cache is being set.
150      */
      public static synchronized void
      setDefaultCache(Cache cache, int dclass) {
          DClass.check(dclass);
          defaultCaches.put(Mnemonic.toInteger(dclass), \
              cache);
155      }

      /**
        * Gets the search path that will be used as the default \
        * by future Lookups.
        * @return The default search path.
160      */
      public static synchronized Name []
      getDefaultSearchPath() {
          return defaultSearchPath;
      }

165      /**
        * Sets the search path to be used as the default by \
        * future Lookups.
        * @param domains The default search path.
        */
170      public static synchronized void
      setDefaultSearchPath(Name [] domains) {
          defaultSearchPath = domains;
      }

175      /**
        * Sets the search path that will be used as the default \
        * by future Lookups.
        * @param domains The default search path.
        * @throws TextParseException A name in the array is not \
        * a valid DNS name.
        */
180      public static synchronized void

```

```

setDefaultSearchPath(String [] domains) throws \
    TextParseException {
    if (domains == null) {
        defaultSearchPath = null;
        return;
185    }
    Name [] newdomains = new Name[domains.length];
    for (int i = 0; i < domains.length; i++)
        newdomains[i] = Name.fromString(domains[i]\
            ], Name.root);
    defaultSearchPath = newdomains;
190 }

protected final void
reset() {
    iterations = 0;
195    foundAlias = false;
    done = false;
    doneCurrent = false;
    aliases = null;
    answers = null;
200    result = -1;
    error = null;
    nxdomain = false;
    badresponse = false;
    badresponse_error = null;
205    networkerror = false;
    timedout = false;
    nametoolong = false;
    referral = false;
    if (temporary_cache)
210        cache.clearCache();
}

/**
 * Create a Lookup object that will find records of the \
    given name, type,
215 * and class. The lookup will use the default cache, \
    resolver, and search
 * path, and look for records that are reasonably \
    credible.
 * @param name The name of the desired records
 * @param type The type of the desired records
 * @param dclass The class of the desired records

```

```

220      * @throws IllegalArgumentException The type is a meta \
      type other than ANY.
      * @see Cache
      * @see Resolver
      * @see Credibility
      * @see Name
225      * @see Type
      * @see DClass
      */
    public
    TorLookup(Name name, int type, int dclass) {
230        Type.check(type);
        DClass.check(dclass);
        if (!Type.isRR(type) && type != Type.ANY)
            throw new IllegalArgumentException("\
                Cannot query for " +
                                                    "meta-\
                types\
                -\
                other\
                -\
                than\
                -\
                ANY\
                ");
235        this.name = name;
        this.type = type;
        this.dclass = dclass;
        synchronized (TorLookup.class) {
            this.resolver = getDefaultResolver();
240            this.searchPath = getDefaultSearchPath();
            this.cache = getDefaultCache(dclass);
        }
        this.credibility = Credibility.NORMAL;
        this.verbose = Options.check("verbose");
245        this.result = -1;
    }

    /**
250     * Create a Lookup object that will find records of the \
        given name and type
        * in the IN class.
        * @param name The name of the desired records
        * @param type The type of the desired records

```

```

    * @throws IllegalArgumentException The type is a meta \
      type other than ANY.
255    * @see #Lookup(Name,int,int)
      */
    public
    TorLookup(Name name, int type) {
        this(name, type, DClass.IN);
260    }

    /**
     * Create a Lookup object that will find records of type \
       A at the given name
     * in the IN class.
265    * @param name The name of the desired records
     * @see #Lookup(Name,int,int)
     */
    public
    TorLookup(Name name) {
270        this(name, Type.A, DClass.IN);
    }

    /**
     * Create a Lookup object that will find records of the \
       given name, type,
275    * and class.
     * @param name The name of the desired records
     * @param type The type of the desired records
     * @param dclass The class of the desired records
     * @throws TextParseException The name is not a valid DNS\
       name
280    * @throws IllegalArgumentException The type is a meta \
       type other than ANY.
     * @see #Lookup(Name,int,int)
     */
    public
    TorLookup(String name, int type, int dclass) throws \
      TextParseException {
285        this(Name.fromString(name), type, dclass);
    }

    /**
     * Create a Lookup object that will find records of the \
       given name and type
290    * in the IN class.
     * @param name The name of the desired records

```

```

    * @param type The type of the desired records
    * @throws TextParseException The name is not a valid DNS\
        name
    * @throws IllegalArgumentException The type is a meta \
        type other than ANY.
295  * @see #Lookup(Name,int,int)
    */
    public
    TorLookup(String name, int type) throws \
        TextParseException {
        this(Name.fromString(name), type, DClass.IN);
300  }

    /**
    * Create a Lookup object that will find records of type \
        A at the given name
    * in the IN class.
305  * @param name The name of the desired records
    * @throws TextParseException The name is not a valid DNS\
        name
    * @see #Lookup(Name,int,int)
    */
    public
310  TorLookup(String name) throws TextParseException {
        this(Name.fromString(name), Type.A, DClass.IN);
    }

    /**
315  * Sets the resolver to use when performing this lookup. \
        This overrides the
    * default value.
    * @param resolver The resolver to use.
    */
    public void
320  setResolver(Resolver resolver) {
        this.resolver = resolver;
    }

    /**
325  * Sets the search path to use when performing this \
        lookup. This overrides the
    * default value.
    * @param domains An array of names containing the search\
        path.
    */

```

```

public void
330 setSearchPath(Name [] domains) {
        this.searchPath = domains;
    }

    /**
335     * Sets the search path to use when performing this \
        lookup. This overrides the
    * default value.
    * @param domains An array of names containing the search\
        path.
    * @throws TextParseException A name in the array is not \
        a valid DNS name.
    */
340 public void
    setSearchPath(String [] domains) throws \
        TextParseException {
        if (domains == null) {
            this.searchPath = null;
            return;
345     }
        Name [] newdomains = new Name[domains.length];
        for (int i = 0; i < domains.length; i++)
            newdomains[i] = Name.fromString(domains[i]\
                ], Name.root);
        this.searchPath = newdomains;
350     }

    /**
    * Sets the cache to use when performing this lookup. \
        This overrides the
    * default value. If the results of this lookup should \
        not be permanently
355     * cached, null can be provided here.
    * @param cache The cache to use.
    */
    public void
    setCache(Cache cache) {
360         if (cache == null) {
            this.cache = new Cache(dclass);
            this.temporary_cache = true;
        } else {
            this.cache = cache;
365         this.temporary_cache = false;
        }
    }

```

```

    }

    /**
370     * Sets the minimum credibility level that will be \
        accepted when performing
    * the lookup. This defaults to Credibility.NORMAL.
    * @param credibility The minimum credibility level.
    */
    public void
375    setCredibility(int credibility) {
        this.credibility = credibility;
    }

    protected void
380    follow(Name name, Name oldname) {
        foundAlias = true;
        badresponse = false;
        networkerror = false;
        timedout = false;
385        nxdomain = false;
        referral = false;
        iterations++;
        if (iterations >= 6 || name.equals(oldname)) {
            result = UNRECOVERABLE;
390            error = "CNAME loop";
            done = true;
            return;
        }
        if (aliases == null)
395            aliases = new ArrayList();
        aliases.add(name);
        lookup(name);
    }

    protected void
400    processResponse(Name name, SetResponse response) {
        if (response.isSuccessful()) {
            RRset [] rrsets = response.answers();
            List l = new ArrayList();
405            Iterator it;
            int i;

            for (i = 0; i < rrsets.length; i++) {
                it = rrsets[i].rrs();
410                while (it.hasNext())

```

```

        l.add(it.next());
    }

    result = SUCCESSFUL;
    answers = (Record []) l.toArray(new \
        Record[l.size()]);
    done = true;
} else if (response.isNXDOMAIN()) {
    nxdomain = true;
    doneCurrent = true;
    if (iterations > 0) {
        result = HOST_NOT_FOUND;
        done = true;
    }
} else if (response.isNXRRSET()) {
    result = TYPE_NOT_FOUND;
    answers = null;
    done = true;
} else if (response.isCNAME()) {
    CNAMERecord cname = response.getCNAME();
    follow(cname.getTarget(), name);
} else if (response.isDNAME()) {
    DNAMERecord dname = response.getDNAME();
    Name newname = null;
    try {
        follow(name.fromDNAME(dname), \
            name);
    } catch (NameTooLongException e) {
        result = UNRECOVERABLE;
        error = "Invalid DNAME target";
        done = true;
    }
} else if (response.isDelegation()) {
    // We shouldn't get a referral. Ignore \
    it.
    referral = true;
}

}

protected void
lookup(Name current) {
    SetResponse sr = cache.lookupRecords(current, \
        type, credibility);
    if (verbose) {

```

```

        System.err.println("lookup_" + current + \
            "_" +
                                Type.string(type));
        System.err.println(sr);
    }
455 processResponse(current, sr);
    if (done || doneCurrent)
        return;

    Record question = Record.newRecord(current, type, \
        dclass);
460 Message query = Message.newQuery(question);
    Message response = null;
    try {
        response = resolver.send(query);
    }
465 catch (IOException e) {
        // A network error occurred. Press on.
        if (e instanceof InterruptedIOException)
            timedout = true;

        else
470             networkerror = true;

        return;
    }
    int rcode = response.getHeader().getRcode();
    if (rcode != Rcode.NOERROR && rcode != Rcode.\
        NXDOMAIN) {
475         // The server we contacted is broken or \
            otherwise unhelpful.
        // Press on.
        badresponse = true;
        badresponse_error = Rcode.string(rcode);
        return;
480    }

    if (!query.getQuestion().equals(response.\
        getQuestion())) {
        // The answer doesn't match the question.\
            That's not good.
        badresponse = true;
485        badresponse_error = "response_does_not_\
            match_query";
        return;
    }

```

```

    sr = cache.addMessage(response);
490    if (sr == null)
        sr = cache.lookupRecords(current, type, \
            credibility);
    if (verbose) {
        System.err.println("queried_" + current + \
            "_" +
                                Type.string(type));
495        System.err.println(sr);
    }
    processResponse(current, sr);
}

500    protected void
    resolve(Name current, Name suffix) {
        doneCurrent = false;
        Name tname = null;
        if (suffix == null)
505            tname = current;
        else {
            try {
                tname = Name.concatenate(current, \
                    suffix);
            }
510            catch (NameTooLongException e) {
                nametoolong = true;
                return;
            }
        }
515        lookup(tname);
    }

    /**
     * Performs the lookup, using the specified Cache, \
        Resolver, and search path.
520     * @return The answers, or null if none are found.
     */
    public Record []
    run() {
525        if (done)
            reset();
        if (name.isAbsolute())
            resolve(name, null);
        else if (searchPath == null)
            resolve(name, Name.root);
    }

```

```

530         else {
            if (name.labels() > 1)
                resolve(name, Name.root);
            if (done)
                return answers;

535         for (int i = 0; i < searchPath.length; i\
            ++){
                resolve(name, searchPath[i]);
                if (done)
                    return answers;
540                 else if (foundAlias)
                    break;
            }
        }
        if (!done) {
545             if (badresponse) {
                result = TRY_AGAIN;
                error = badresponse_error;
                done = true;
            } else if (timedout) {
550                 result = TRY_AGAIN;
                error = "timed_out";
                done = true;
            } else if (networkerror) {
                result = TRY_AGAIN;
555                 error = "network_error";
                done = true;
            } else if (nxdomain) {
                result = HOST_NOT_FOUND;
                done = true;
560             } else if (referral) {
                result = UNRECOVERABLE;
                error = "referral";
                done = true;
            } else if (nametoolong) {
565                 result = UNRECOVERABLE;
                error = "name_too_long";
                done = true;
            }
        }
570     return answers;
}

```

protected void

```

checkDone() {
575     if (done && result != -1)
        return;
    StringBuffer sb = new StringBuffer("Lookup_of_" + \
        name + "_");
    if (dclass != DClass.IN)
        sb.append(DClass.string(dclass) + "_");
580     sb.append(Type.string(type) + "_isn't done");
    throw new IllegalStateException(sb.toString());
}

/**
585  * Returns the answers from the lookup.
  * @return The answers, or null if none are found.
  * @throws IllegalStateException The lookup has not \
    completed.
  */
public Record []
590 getAnswers() {
    checkDone();
    return answers;
}

/**
595  * Returns all known aliases for this name. Whenever a \
    CNAME/DNAME is
  * followed, an alias is added to this array. The last \
    element in this
  * array will be the owner name for records in the answer \
    , if there are any.
  * @return The aliases.
600  * @throws IllegalStateException The lookup has not \
    completed.
  */
public Name []
getAliases() {
    checkDone();
605     if (aliases == null)
        return noAliases;
    return (Name []) aliases.toArray(new Name[aliases \
        .size()]);
}

/**
610  * Returns the result code of the lookup.

```

```

        * @return The result code, which can be SUCCESSFUL, \
          UNRECOVERABLE, TRY_AGAIN,
        * HOST_NOT_FOUND, or TYPE_NOT_FOUND.
        * @throws IllegalStateException The lookup has not \
          completed.
615    */
    public int
    getResult() {
        checkDone();
        return result;
620    }

    /**
     * Returns an error string describing the result code of \
       this lookup.
     * @return A string, which may either directly correspond\
       the result code
625    * or be more specific.
     * @throws IllegalStateException The lookup has not \
       completed.
    */
    public String
    getErrorString() {
630        checkDone();
        if (error != null)
            return error;
        switch (result) {
            case SUCCESSFUL:        return "\
                                   successful";
635            case UNRECOVERABLE:    return "\
                                   unrecoverable_error";
            case TRY_AGAIN:        return "try_again\
                                   ";
            case HOST_NOT_FOUND:    return "host_not_\
                                   found";
            case TYPE_NOT_FOUND:    return "type_not_\
                                   found";
        }
640        throw new IllegalStateException("unknown_result")\
            ;
    }

    public static void
645    printAnswer(String name, TorLookup lookup) {

```

```

        System.out.print(name + ":" );
        int result = lookup.getResult();
        if (result != Lookup.SUCCESSFUL)
            System.out.print("_" + lookup.\
                getErrorString());
650    System.out.println();
        Name [] aliases = lookup.getAliases();
        if (aliases.length > 0) {
            System.out.print("#_aliases:_");
            for (int i = 0; i < aliases.length; i++) \
                {
655                System.out.print(aliases[i]);
                    if (i < aliases.length - 1)
                        System.out.print("_");
                }
            System.out.println();
660        }
        if (lookup.getResult() == Lookup.SUCCESSFUL) {
            Record [] answers = lookup.getAnswers();
            for (int i = 0; i < answers.length; i++)
                System.out.println(answers[i]);
665        }
    }

    public static void
main(String [] args) throws Exception {
670        int type = Type.A;
        int start = 0;
        Options.set("Verbose");
        if (args.length > 2 && args[0].equals("-t")) {
            type = Type.value(args[1]);
            if (type < 0)
675                throw new \
                    IllegalArgumentException("\
                        invalid_type");
            start = 2;
        }
        for (int i = start; i < args.length; i++) {
680            TorLookup l = new TorLookup(args[i], type\
                );
            l.run();
            printAnswer(args[i], l);
        }
    }
685

```

```
}
```

B.3.3 TorResolver.java

This java code takes care of preparing and interpreting a lookup operation.

```
/**
 *
 */
4 package TorLib;

import java.io.IOException;
import java.net.InetAddress;
import java.net.InetSocketAddress;
9 import java.net.UnknownHostException;
import java.util.Iterator;
import java.util.List;

import org.xbill.DNS.Flags;
14 import org.xbill.DNS.Header;
import org.xbill.DNS.Message;
import org.xbill.DNS.Name;
import org.xbill.DNS.OPTRecord;
import org.xbill.DNS Opcode;
19 import org.xbill.DNS.Options;
import org.xbill.DNS.Rcode;
import org.xbill.DNS.Record;
import org.xbill.DNS.Resolver;
import org.xbill.DNS.ResolverListener;
24 import org.xbill.DNS.Section;
import org.xbill.DNS.TCPClient;
import org.xbill.DNS.TSIG;
import org.xbill.DNS.Type;
import org.xbill.DNS.UDPClient;
29 import org.xbill.DNS.WireParseException;
import org.xbill.DNS.ZoneTransferException;
import org.xbill.DNS.ZoneTransferIn;

/**
34 * @author foley
 *
 */
public class TorResolver implements Resolver {

39     /** The default port to send queries to */
```

```

public static final int DEFAULT_PORT = 53;

/** The default EDNS payload size */
public static final int DEFAULT_EDNS_PAYLOAD_SIZE = 1280;
44
/** Changed address and localAddress to protected so \
    subclass can occur */
protected InetSocketAddress address;
protected InetSocketAddress localAddress;
protected boolean useTCP, ignoreTruncation;
49 protected OPTRecord queryOPT;
protected TSIG tsig;
protected int timeoutValue = 10 * 1000;

protected static final short DEFAULT_UDPSIZE = 512;
54
protected static String defaultResolver = "bitsy.mit.edu"\
    ;
protected static int uniqueID = 0;

/**
59  * This method constructs the resolver to be used.
 * @param hostname - server to be queried
 * @throws UnknownHostException
 */

64 public TorResolver(String hostname) throws \
    UnknownHostException {
    useTCP = true;
    String ipaddr = TorLib.TorResolve(hostname);
    System.err.println("TorResolve:_" + ipaddr);

69     address = new InetSocketAddress(ipaddr, \
        DEFAULT_PORT);
    System.err.println("Sending_to_" +
        address.getAddress().\
            getHostAddress() +
            ":" + address.getPort());

74 }

/**
     * @throws UnknownHostException
     */
79 public TorResolver() throws UnknownHostException {

```

```

        this(defaultResolver);
    }

    /**
84      * Sends a message to a single server and waits for a \
        response. No checking
        * is done to ensure that the response is associated with\
        the query.
        * @param query The query to send.
        * @return The response.
        * @throws IOException An error occurred while sending or\
        receiving.
89      */
    public Message
    send(Message query) throws IOException {
        if (Options.check("verbose"))
            System.err.println("Sending to " +
94                address.getAddress().\
                    getHostAddress() +
                    ":" + address.getPort\
                    ());

        if (query.getHeader().getOpcode() == Opcode.QUERY\
            ) {
            Record question = query.getQuestion();
            if (question != null && question.getType\
                () == Type.AXFR)
                return sendAXFR(query);
        }

        query = (Message) query.clone();
        applyEDNS(query);
104        if (tsig != null)
            tsig.apply(query, null);

        byte [] out = query.toWire(Message.MAXLENGTH);
109        int udpSize = maxUDPSize(query);
        boolean tcp = false;
        long endTime = System.currentTimeMillis() + \
            timeoutValue;

        do {
114            byte [] in;

            // always use TCP with Tor

```

```

//          if (useTCP || out.length > udpSize)
//              tcp = true;
119 //      if (tcp) {
//          in = TorTCPClient.sendrecv(\
//              localAddress, address, out,
//                  endTime);
//      }
//      else
124 //          in = UDPClient.sendrecv(\
//              localAddress, address, out,
//                  udpSize, \
//                      endTime);

/*
 * Check that the response is long enough\
 */
129 /*
if (in.length < Header.LENGTH) {
    throw new WireParseException("\
        invalid DNS header _" +
            "too\
            _\
            short\
            "\
            )\
            ;
}
134 /*
 * Check that the response ID matches the\
 * query ID. We want
 * to check this before actually parsing \
 * the message, so that
 * if there's a malformed response that's\
 * not ours, it
 * doesn't confuse us.
139 */
int id = ((in[0] & 0xFF) << 8) + (in[1] &\
    0xFF);
int qid = query.getHeader().getID();
if (id != qid) {
    String error = "invalid message_\
        id: _expected_" + qid +
144         " _; _got _id_" + id;

    if (tcp) {

```

```

                                throw new \
                                    WireParseException(\
                                        error);
                                } else {
                                    if (Options.check("\
                                        verbose")) {
149                                     System.err.\
                                         println(error)\
                                             ;
                                    }
                                    continue;
                                }
                            }
154         Message response = parseMessage(in);
        verifyTSIG(query, response, in, tsig);
        if (!tcp && !ignoreTruncation &&
            response.getHeader().getFlag(Flags.TC\
                ))
        {
159             tcp = true;
            continue;
        }
        return response;
    } while (true);
164 }

InetSocketAddress
getAddress() {
    return address;
169 }

/** Sets the default host (initially localhost) to query \
    */
public static void
setDefaultResolver(String hostname) {
174     defaultResolver = hostname;
}

public void
setPort(int port) {
179     address = new InetSocketAddress(address.\
        getAddress(), port);
}

/**

```

```

    * Sets the address of the server to communicate with.
184    * @param addr The address of the DNS server
    */
    public void
    setAddress(InetSocketAddress addr) {
        address = addr;
189    }

    /**
    * Sets the address of the server to communicate with (on\
        the default
    * DNS port)
194    * @param addr The address of the DNS server
    */
    public void
    setAddress(InetAddress addr) {
        address = new InetSocketAddress(addr, address.\
            getPort());
199    }

    /**
    * Sets the local address to bind to when sending \
        messages.
    * @param addr The local address to send messages from.
204    */
    public void
    setLocalAddress(InetSocketAddress addr) {
        localAddress = addr;
    }
209

    /**
    * Sets the local address to bind to when sending \
        messages. A random port
    * will be used.
    * @param addr The local address to send messages from.
214    */
    public void
    setLocalAddress(InetAddress addr) {
        localAddress = new InetSocketAddress(addr, 0);
    }
219

    public void
    setTCP(boolean flag) {
        this.useTCP = flag;
    }

```

```

224     public void
        setIgnoreTruncation(boolean flag) {
            this.ignoreTruncation = flag;
        }

229     public void
        setEDNS(int level, int payloadSize, int flags, List \
            options) {
            if (level != 0 && level != -1)
                throw new IllegalArgumentException("\
                    invalid EDNS level - " +
234                                     "must be\
                                                0\
                                                or\
                                                -1" \
                                                );

            if (payloadSize == 0)
                payloadSize = DEFAULT_EDNS_PAYLOAD_SIZE;
            queryOPT = new OPTRecord(payloadSize, 0, level, \
                flags, options);
        }

239     public void
        setEDNS(int level) {
            setEDNS(level, 0, 0, null);
        }

244     public void
        setTSIGKey(TSIG key) {
            tsig = key;
        }

249     public void
        setTSIGKey(Name name, byte [] key) {
            tsig = new TSIG(name, key);
        }

254     public void
        setTSIGKey(String name, String key) {
            tsig = new TSIG(name, key);
        }

259     TSIG

```

```

getTSIGKey() {
    return tsig;
}

264 public void
setTimeout(int secs) {
    timeoutValue = secs * 1000;
}

269 int
getTimeout() {
    return timeoutValue / 1000;
}

274 protected Message
parseMessage(byte [] b) throws WireParseException {
    try {
        return (new Message(b));
    }
    279 catch (IOException e) {
        if (Options.check("verbose"))
            e.printStackTrace();
        if (!(e instanceof WireParseException))
            284 e = new WireParseException("Error\
                _parsing_message");
        throw (WireParseException) e;
    }
}

289 protected void
verifyTSIG(Message query, Message response, byte [] b, \
    TSIG tsig) {
    if (tsig == null)
        return;
    int error = tsig.verify(response, b, query.\
        getTSIG());
    294 if (error == Rcode.NOERROR)
        response.tsigState = Message.\
            TSIG_VERIFIED;
    else
        response.tsigState = Message.TSIG_FAILED;
    if (Options.check("verbose"))
        299 System.err.println("TSIG_verify:_ " + \
            Rcode.string(error));
}

```

```

    protected void
    applyEDNS(Message query) {
304         if (queryOPT == null || query.getOPT() != null)
            return;
        query.addRecord(queryOPT, Section.ADDITIONAL);
    }

309    protected int
    maxUDPSize(Message query) {
        OPTRecord opt = query.getOPT();
        if (opt == null)
            return DEFAULT_UDPSIZE;
314        else
            return opt.getPayloadSize();
    }

    /**
319     * Asynchronously sends a message to a single server, \
        registering a listener
    * to receive a callback on success or exception. \
        Multiple asynchronous
    * lookups can be performed in parallel. Since the \
        callback may be invoked
    * before the function returns, external synchronization \
        is necessary.
    * @param query The query to send
324     * @param listener The object containing the callbacks.
    * @return An identifier, which is also a parameter in \
        the callback
    */
    public Object
    sendAsync(final Message query, final ResolverListener \
        listener) {
329        System.err.println("Something_wants_TorResolver.\
            sendAsync");
        return(null);
    }

    protected Message
334    sendAXFR(Message query) throws IOException {
        Name qname = query.getQuestion().getName();
        ZoneTransferIn xfrin = ZoneTransferIn.newAXFR(\
            qname, address, tsig);
        xfrin.setTimeout(getTimeout());
    }

```

```

339         try {
                xfrin.run();
            }
            catch (ZoneTransferException e) {
                throw new WireParseException(e.getMessage\
                    ());
            }
344         List records = xfrin.getAXFR();
        Message response = new Message(query.getHeader().\
            getID());
        response.getHeader().setFlag(Flags.AA);
        response.getHeader().setFlag(Flags.QR);
        response.addRecord(query.getQuestion(), Section.\
            QUESTION);
349         Iterator it = records.iterator();
        while (it.hasNext())
            response.addRecord((Record)it.next(), \
                Section.ANSWER);
        return response;
    }

354

    /**
     * @param args
     */
359     public static void main(String[] args) {
        // lets test if the resolver works

    }

364 }

```

B.3.4 TorSocketChannel.java

This is a general purpose socket implementation that uses the Tor SOCKS4a interface for anonymization.

```

package TorLib;

import java.io.IOException;
import java.net.Socket;
5 import java.net.SocketAddress;
import java.nio.ByteBuffer;
import java.nio.channels.SocketChannel;

```

```
import java.nio.channels.spi.SelectorProvider;

10 public class TorSocketChannel extends SocketChannel {

    protected TorSocketChannel(SelectorProvider provider) {
        super(provider);
15        // TODO Auto-generated constructor stub
    }

    /**
     * @param args
20    */
    public static void main(String[] args) {
        // TODO Auto-generated method stub

    }

25    public Socket socket() {
        // TODO Auto-generated method stub
        return null;
    }

30    public boolean isConnected() {
        // TODO Auto-generated method stub
        return false;
    }

35    public boolean isConnectionPending() {
        // TODO Auto-generated method stub
        return false;
    }

40    public boolean connect(SocketAddress remote) throws \
        IOException {
        // TODO Auto-generated method stub
        return false;
    }

45    public boolean finishConnect() throws IOException {
        // TODO Auto-generated method stub
        return false;
    }

50    public int read(ByteBuffer dst) throws IOException {
```

```

        // TODO Auto-generated method stub
        return 0;
    }

55     public long read(ByteBuffer[] dsts, int offset, int \
        length) throws IOException {
        // TODO Auto-generated method stub
        return 0;
    }

60     public int write(ByteBuffer src) throws IOException {
        // TODO Auto-generated method stub
        return 0;
    }

65     public long write(ByteBuffer[] srcs, int offset, int \
        length) throws IOException {
        // TODO Auto-generated method stub
        return 0;
    }

70     protected void implCloseSelectableChannel() throws \
        IOException {
        // TODO Auto-generated method stub

    }

75     protected void implConfigureBlocking(boolean block) \
        throws IOException {
        // TODO Auto-generated method stub

    }

80 }

```

B.3.5 TorTCPClient.java

This is a general purpose TCP client implementation that uses the Tor SOCKS4a interface for anonymization.

```

package TorLib;

import java.io.DataInputStream;
4 import java.io.DataOutputStream;
import java.io.EOFException;

```

```

import java.io.IOException;
import java.net.InetSocketAddress;
import java.net.Socket;
9 import java.net.SocketAddress;
import java.nio.ByteBuffer;
import java.nio.channels.SelectionKey;
import java.nio.channels.SocketChannel;

14 import org.xbill.DNS.Options;
import org.xbill.DNS.utils.hexdump;

public class TorTCPClient {
    SocketAddress sockProxyAddr;
19    SocketAddress sockDestAddr;
    Socket torSock;

    public TorTCPClient(long endTime) throws IOException {
        // endtime should be the timeout, but I'm lazy
24    }

    protected long endTime;
    protected SelectionKey key;

29    static protected void
    verboseLog(String prefix, byte [] data) {
        if (Options.check("verbosemsg"))
            System.err.println(hexdump.dump(prefix, \
                data));
    }

34    public void
    cleanup() throws IOException {
    }

39    /**
     * This method does the switcheroo on the socket We bind \
        to the proxy
     * address instead of the destination
     */
44    public void bind(SocketAddress addr) throws IOException {
        sockProxyAddr = new InetSocketAddress(TorLib.\
            proxyAddr,
                TorLib.proxyPort);
        sockDestAddr = addr;

```

```

        System.err.println("TorTCPClient:_bind");
49     }

    public void connect(SocketAddress addr) throws \
        IOException {
        /** Now we send Tor Init */
        try {
54             InetSocketAddress saddr = (InetSocketAddress)\
                addr;
                torSock = TorLib.TorSocket(saddr.\
                    getHostName(), saddr.getPort());
        }
        catch(Exception e) {e.printStackTrace();}
    }

59     public void
    send(byte [] data) throws IOException {
        DataOutputStream out = new DataOutputStream(\
            torSock.getOutputStream());
        verboseLog("TCP_write", data);
64         byte [] lengthArray = new byte[2];
        lengthArray[0] = (byte)(data.length >>> 8);
        lengthArray[1] = (byte)(data.length & 0xFF);
        ByteBuffer [] buffers = new ByteBuffer[2];

69         try {
            out.write(lengthArray);
            out.write(data, 0, data.\
                length);
        }
        catch(Exception e) {e.printStackTrace();}

74     }

    protected byte []
    _recv(int length) throws IOException {
        DataInputStream in = new DataInputStream(torSock.\
            getInputStream());

79         int nrecvd = 0;
        byte [] data = new byte[length];
        try {
            in.read(data, 0, length);

84         }
        catch(Exception e) {e.printStackTrace();}
    }

```

```

        return data;
    }

89
    public byte []
    recv() throws IOException {
        byte [] buf = _recv(2);
        int length = ((buf[0] & 0xFF) << 8) + (buf[1] & 0\
94
            xFF);
        byte [] data = _recv(length);
        verboseLog("TCP_read", data);
        return data;
    }

99
    public static byte []
    sendrecv(SocketAddress local, SocketAddress remote, byte \
        [] data, long endTime)
    throws IOException
    {
        TorTCPClient client = new TorTCPClient(endTime);
104
        try {
            if (local != null)
                client.bind(local);
            client.connect(remote);
            client.send(data);
109
            return client.recv();
        }
        finally {
            client.cleanup();
        }
114
    }

    public static byte []
    sendrecv(SocketAddress addr, byte [] data, long endTime) \
        throws IOException {
        return sendrecv(null, addr, data, endTime);
119
    }

    /**
     * @param args
124
    */
    public static void main(String[] args) {
        // TODO Auto-generated method stub

    }

```

129
}

Appendix C

Terminology

Active tag: a tag with an internal power source, usually with an internal RF transmitter allowing a long read range, Page 22

Agile reader: a reader capable of operating in multiple frequencies, encoding standards and/or protocols, Page 27

Anti-counterfeiting: a mechanism to verify the authenticity of a tag and its associated object, Page 103

Application Programming Interface: a specification for how an application can interface with a code or networking library., Page 171

Asset: An asset is an item, digital or physical, that has a measurable value. Disclosure or corruption of digital assets would often result in a loss., Page 83

authentication chain: DNSSEC's [dnssb] mechanism for following a set of chained cryptographic signatures to determine authority, Page 66

Authentication: the process of verifying the claimed identity of an entity, Page 102

Authorization: granting permission to access a given resource or information, or the procedure for granting said access, Page 163

Back-scatter: This is a method for transmitting information in RFID. The tag has an antenna designed to reflect a specific RF frequency back to the originating source if the impedance is set to a specific value. The tag can change the reflection properties to modulate data by changing the impedance., Page 20

Cloned tag: a perfect copy of another tag, at least to the process being examined, Page 39

- Cover-coding: a method of obscuring transmission from a tag to a reader, in which the tag sends the reader a random challenge, the reader XORs the challenge with the data to be returned and the tag decodes the data by XORing it with the challenge once again. [EPC05a], Page 35
- Dictionary attack: pre-computing all possible keys with a known plain-text to determine which key was used, Page 97
- Dynamic Host Configuration Protocol (DHCP): network protocol based upon BOOTP that allows a computer to request a temporary IP address and other configuration information., Page 172
- Eavesdropping: listening to communications between two parties, possibly with malicious intent (also known as “skimming”), Page 47
- Elliptic Curve Cryptography (ECC): public key cryptography that uses elliptic curve calculations instead of modular arithmetic, Page 48
- Encryption: a cryptographic transformation of data that conceals it such that only entities with a specific piece of information are able to reverse the transformation, Page 24
- Federation: a security model with localized authentication used along with limited trust between domains of control allowing an authenticated entity in one domain to gain access to another domain, Page 70
- Friendly fire: being shot at by your allies because you’ve been mistaken for an enemy, Page 85
- Functional Requirement (FR): a system design concept, which describes the necessary specifications that a given design must satisfy, Page 92
- Inductive coupling: Passive tags use inductive coupling to transmit information. The tag’s antenna coil is switched on and off to modulate the power draw upon the reader’s transmission antenna. The changes in power are demodulated into the original data., Page 34
- Jamming: decreasing the signal-to-noise ratio of a communication channel such that it is extremely difficult to extract information from it, Page 45
- Locking: changing the state of a tag such that some operations act differently (for example, locked tags will not answer certain types of queries, or simply not allow modification to the data stored on the tag), Page 35

- Meet in the middle attack: an attack that trades storage space for speed, decreasing the search time for a key by encrypting a known plaintext with all possible keys and decrypting a known ciphertext with all possible keys to find the individual round keys in a chained encryption system., Page 113
- Mix-net: a large network of servers, which take packets and randomly (or pseudo-randomly such that a return path is possible) route them through the network to obscure their origin and path, Page 168
- Non-repudiation: a mechanism for proving a party's involvement in a process or transaction (for example, the payer's signature on a check), Page 140
- Nonce: a random or sequential token that is guaranteed to only be used once, Page 41
- NP-hard problems: NP stands for "Non-deterministic Polynomial time". NP problems are solvable in polynomial time on a non-deterministic Turing machine. More intrinsically, NP problems can be verified in polynomial time. NP-hard refers to problems that are at least as hard as any decision problem in the NP. More informally, NP-hard problems are problems that require significant computing power to solve., Page 49
- Passive tag: Passive tags have no constant internal power source and are therefore powered by the reader. Most designs have a small capacitor to temporarily store energy captured from the reader's RF field. The read range on this type of tag is fairly short, depending upon frequency, size of antenna, and sensitivity of the reader. Most tags of this sort use inductive coupling as a power source and transmission mechanism., Page 34
- Personal Information Carrier: Personal Digital Assistant (PDA) that has been ruggedized for combat and modified to use applications for medical and operations planning, Page 69
- POS: Point of Sale. Usually a cash register and associated equipment., Page 130
- Pre-resolver: the ONS middle-ware that translates RFID raw data into ONS parsed names, Page 61
- Pseudo-Random Number Generator (PRNG): This mechanism uses complex mathematical functions and saves state to emulate truly random numbers. Unfortunately, the functions are deterministic, so ensuring that said state or "seed", is saved to prevent duplication is important., Page 48
- Radio Frequency Identification (RFID): the use of radio frequency energy to gain an identifier associated with an entity, Page 19

- Random Number Generator (RNG): a mechanism for generating random numbers, often implemented as a Pseudo-Random Number Generator (PRNG), Page 48
- Random oracle: a theoretical black box that maps every possible query to a random response from the domain of its outputs, Page 114
- Reader: a device used to access an RFID tag and write information to it (also known as an “interrogator”), Page 28
- Savant: a minimalist server that administers a small cloud of entities, processes information from them, and will answer queries about the information using XML-RPC or SOAP, Page 27
- Semi-passive tag *a.k.a.* semi-active tag: a tag that has an internal power source for powering the controller, but whose transceiver uses the reader’s RF power to communicate, often in a manner such as back-scatter, Page 34
- Service: a computer system with the ability to store and process information over a network, Page 19
- Silicon Physical Random Function (PUF): a chip design that amplifies the small inaccuracies in manufacturing to produce a random number or function, Page 39
- Single Sign-On (SSO): an authentication system capability that allows an entity to use an initial authentication cycle to be applied to other accessible services without a new authentication cycle, Page 71
- Skimming: a passive attack in which an attacker monitors readers and/or tags to gather information that will be replayed later (also known as “eavesdropping”), Page 39
- Spoofing: emulating a tag or reader with the intent of gaining access to unauthorized information, Page 40
- Tag kill: a method by which a tag is silenced, ranging from simply physically destroying the tag to modifying its internal programming such that it stops answering queries, Page 36
- Visibility: the capability to locate an item and any associated information quickly and easily, Page 85

Index

- “My Hovercraft is full of eels”, 105
- Advanced Encryption Standard, 48
- anti-counterfeiting
 - for Money, 20
 - introduction, 103–106
- asset, 83
- authentication, 102
 - chain, 66
 - survey of RFID mechanisms, 51
 - tag-to-reader, 47
- authorization, 163
- automated toll taking, 20
- back-scatter, 20, 24, 35
- bar-code, 19
- Big Brother, 23
- binary tree-walking, 167
- blocker
 - soft, 46
 - tag, 44–46
- booster bags, 47
- Bovine Spongiform Encephalopathy, 22
- Confirmed Kill, 30, 152–154, 179
- constellation, 30
- cover-coding, 35, 41
- cryptography
 - elliptic curve, 47
 - key predistribution, 75–76
 - public-key, 32
 - symmetric, 32
- Data Guardian, 68
- DDHV, *see* Du-Deng-Han-Varshney
- Deadline timer, 148–150, 163, 179
- decoupled verification, 108
- delegation tree, 69
- demesnes, 62–64, 174
- DES, 48, *see* Digital Encryption Standard
- Digital Encryption Standard, 48
- Digital Signal Transponder, 101
- DNS, *see* Domain Name System, 29, *see* Domain Name System
- Domain Name System, 25, 54, 55
- Du-Deng-Han-Varshney, 75
- Duracell, 101
- DVD content scrambling system(DVDCSS), 102
- Dynamic Hash Table, 65
- EAS, 21, 47
- eavesdropping, 47
- EG, *see* Eschenaur-Gligor
- Electronic
 - Article Surveillance, 19
 - Product Code, 3, 25, 33
- electronic possession, 160
- end-to-end, 30
- Enterprise Privacy Authorization Language, 70
- EPAL, *see* Enterprise Privacy Authorization Language
- ePassport, 29
- EPC, 19, 25, 27, 29, 30, 32–38, 40, 60–62
 - IS, 29, 60, 62, 66, 165, 168
 - Class 1 Generation 2, 30, 33–39

- states, 35–36
- EPC-IS, 29
- EPCglobal, 27, 34, 35, 60
 - Tag Data Standard, 34, 60
- ephemeral memory, 150
- Eschenauro-Gligor, 76
- exploding chips, *see* gadolinium nitrate
- Exponential Ramp-up, 3, 156–160, 179
- eXtensible Markup Language, 25
- EZ-pass, *see* automated toll taking
- fake IDs, 172–173
- Faraday cage, 167
- Fastlane, 101
- federation, 70–73
- firewall, 174
- gadolinium nitrate, 39
- Galois Field, 49
- gaming chips, 21, 101
- Gen2, 35, *see* EPC Class 1 Generation 2, 36, 38, 41
- GF, *see* Galois Field
- GlueSniffer, 172
- grenade timer, 148
- GS1, 27
- hash functions
 - RFID, 109–116
- hash lock, 32, 41
- hash locks, 41–43
- HB, *see* Hopper-Blum
- HB+, 50–51
- Hesiod, 57–59
- Homeland Security, 29
- hood, *see* neighborhood
- Hopper-Blum, 49
- house arrest, 21
- Identification Friend or Foe, 24
- IFF, *see* Identification Friend or Foe
- inductive coupling, 34
- information leakage, 3
- ISO 10374.2, 21
- ISO/IEC 18000, 33
- jamming, 45
 - radio frequency, 45
- JanusTag, 3, 30, 41, 98, 102–103, 177
- Jekyll and Hyde, 102
- Kerberos, 71–73
- Kill Resistance, 178
- killer
 - serial, 144
- killing, 144
 - machine gun, 144
- Laboratory for Catching Crackers, 173
- leakage
 - of secrets, 30
- Liberty, 71
- livestock, 22–23
- lost children, 22
- Machine Readable Travel Document, 29
- machine-gun
 - reader, 147
- Mad Cow Disease, *see* Bovine Spongiform Encephalopathy
- Man-in-the-Middle, 166–167
- mapping database, 102
- mass kill, 3, 144–160
- meat packing, 23
- Meet-in-the-middle, 113
- metaID, *see* hash lock
- microwave oven, 28
- MIT
 - Auto-ID Center, 27, 33
 - Auto-ID Labs, 27
 - card reverse-engineering, 40
 - DISC, 27, 28
 - Distributed Intelligent Systems Center, 24, 33
 - Laboratory for Manufacturing and Productivity, 24
- NAT, *see* Network Address Translation

- Near Field Communications, 87
- neighborhood, 147
- Neighborhood Watch, 3, 98, 147, 178
- Network Address Translation, 174
- NFC, *see* Near Field Communications
- noisy Bloom filter, 43
- notation, 31

- Object Name System, 3, 25, 29, 60–64
- ONS, *see* Object Name System, 27, 29, 64, 165–171
 - Leakage, 165–167
 - Misdirection, 171–173
 - star-records, 62
 - Tor-, 168–171
- oracle, 114

- P3P, 67, 68, 70
- pallets, 21
- Personal Information Carrier, 68
- PGP, 147
- PIC, *see* Personal Information Carrier
- Platform for Privacy Practices, *see* P3P
- PML, 25, 29
- Point of Sale, 19, 130
- poker chips, *see* gaming chips
- POS, *see* Point of Sale
- PRNG, *see* Pseudo-Random Number Generator
- Product Markup Language, *see* PML
- Project Mayhem, 177
- Proof of Life, 160–161, 164
- Pseudo-Random Number Generator, 48
 - Fortuna, 116
- PUF, *see* Silicon Physical Random Function
- puzzles, 155–156

- RADAR, *see* Radio Detection and Ranging
- Radio Detection and Ranging, 24
- Radio Frequency Identification, 3
- Random Number Generator, 48

- RAT, *see* RFID Acceptor Tag
- reader, 20, 27–30, 32
 - agile, 27
- Reader Misdirection, 30, 179
- resolution, 53–66
 - host tables, 54
- resurrecting ducklings, 75
- RFID, *see* Radio Frequency Identification
 - 1-bit tags, 21
 - Acceptor Tag, 69
 - avoidance, 46
 - barriers, 46
 - certification, 47
 - destruction, 46
 - freight containers, 20
 - History, 23–27
 - at MIT, 24–27
 - in libraries, 40
 - physical countermeasures, 46–47
 - recoding, 40, 41
 - viruses, 40
- Rijndael, *see* Advanced Encryption Standard
- RNG, *see* Random Number Generator
- Robert Paulson, 88–89, 187–188

- Savant, 27
- script kiddies, 172
- Secret Service, 28
- seed, 48
- self-defense, 3
- Service, 19
- Signal-to-Noise Ratio, 173
- Silicon Physical Random Function, 39
- Simple Query Language, 70
- Single Sign-On, 71, 72
- SIREN, 70
- SpeedPass, 101
- SQL, *see* Simple Query Language
- SSL, 106
- SSO, *see* Single Sign-On
- START button, 25

- Stay of Execution, 150–152, 179
- steganography, 173
- tag
 - active, 22
 - cloned, 39
 - kill, 36
 - locking, 35
 - microwaved, 46
 - passive, 34
 - Privacy Agent, 46
 - protection, 33–53
 - semi-active, 34
 - semi-passive, 34
 - subdermal, 20
- Tag Directive, 161–164
- Tag Folio, 164
- TagCheck, 3, 30, 98, 107–117, 178
 - hash-based, 107–109
 - symmetric, 116–117
- TagDirective, 3, 31, 98, 179
- TagFolio, 31, 98, 128–141, 178
- tags
 - sticky, 173
- Temporary Reprieve, 148
- The onion router, 62
- threat analysis, 184–188
- TinFoil, 3, 180–188
- TinFoil, 99
- TinFoil, 31
- Tor, *see* The onion router, 168–170
- tor-resolve, 170–171
- TorONS, 31, 66, 98, 168–171, 180
- UbiComp, 68
- UCC, 27
- Uniform Resource Indicator, 61
- Universal Product Code, 25
- URI
 - see* Uniform Resource Indicator, 61
- Vegas S, 21
- Verichip, 20
- Virtual Private Network, 64
- VPN, *see* Virtual Private Network
- Watchdog Tag, 69
- web of trust, 73–75, 147
- XML, *see* eXtensible Markup Language, 29

Bibliography

- [1Sya] The 1sync homepage. Available from: www.1sync.org.
- [1Syb] Inc. 1Sync. Price and promotion synchronization. Previously UCC-Net.org, sponsored by GS1US.
- [260] Secret service codenames. Available from: <http://www.2600.com/secret/more/codes.html>.
- [AHK⁺03] Paul Ashley, Satoshi Hada, Günter Karjoth, Calvin Powers, and Matthias Schunter. Enterprise privacy authorization language, November 2003. Available from: www.w3.org/Submission/2003/SUBM-EPAL-20031110/.
- [AL97] Paul Albitz and Cricket Liu. *DNS and BIND*. O'Reilly & Associates, Inc., Sebastopol, California, second edition, 1997.
- [Ame06] 2006. American Express Website. Available from: home.americanexpress.com.
- [AO05] G. Avoine and P. Oechslin. A scalable and provably secure hash based RFID protocol. In *International Workshop on Pervasive Computing and Communication Security – PerSec 2005*, Kauai Island, Hawaii, March 2005. IEEE Computer Society Press.
- [Ass] American Library Association. The USA PATRIOT act. Information on how the PATRIOT Act pertains to libraries. Available from: www.ala.org/ala/washoff/WOissues/civilliberties/theusapatriotact/usapatriotact.htm.
- [Aut] AutoID labs website. Available from: www.autoidlabs.org.
- [Baa06] Mark Baard. Retail-safe RFID unveiled. *Wired Magazine*, May 2006. Available from: wired.com/news/technology/0,70793-0.html.
- [Bel95] Steven M. Bellovin. Using the domain name system for system break-ins. *Proceedings of the Fifth Usenix UNIX Security Symposium*, June 1995.

- [BF04] Matt Bishop and Deborah A. Frinke. Guarding the castle keep: Teaching with the fortress metaphor. *IEEE Security & Privacy*, 2:69–72, May-June 2004.
- [BGK⁺06] L. Batina, J. Guarjardo, T. Kerins, N. Menthens, P. Tuyls, and I. Verbauwhede. Public-key cryptography for RFID-tags. In Sandra Dominikus, editor, *Workshop on RFID Security 2006 (RFIDSEC '06)*, pages 61–76, July 2006.
- [BJ06] D. Bailey and A. Juels. Shoehorning security into the EPC standard. Manuscript in submission, January 2006.
- [Blo84] Rolf Blom. An optimal class of symmetric key generation systems. In *EUROCRYPT*, pages 335–338, 1984.
- [BM04] Jürgen Bohn and Friedemann Mattern. Super-Distributed RFID Tag Infrastructures. In Panos Markopoulos, Berry Eggen, Emile Aarts, and James Crowley, editors, *Proc. 2nd European Symposium on Ambient Intelligence (EUSAI 2004)*, number 3295 in LNCS, pages 1–12, Eindhoven, The Netherlands, November 2004. Springer-Verlag. Available from: citeseer.ist.psu.edu/bohn04superdistributed.html.
- [Bro99] Roberta Brody. Consequences of electric profiling. *IEEE Technology and Society*, 18(1):20–27, Spring 1999.
- [Bro01] David Brock. The electronic product code(EPC): A naming scheme for physical objects. *AutoID Center*, January 2001. MIT-AUTOID-WH-002.
- [BRS02] J. Black, P. Rogaway, and T. Shrimpton. Black-box analysis of the block-cipher-based hash function constructions from pcv. In *Advances in Cryptology CRYPTO'02*. Springer-Verlag, 2002. Available from: citeseer.ist.psu.edu/article/black02blackbox.html.
- [BRSG06] Stephen Bono, Aviel Ruben, Adam Stubblefield, and Matthew Green. Security through legality. *Communications of the ACM*, 49(6):41–43, June 2006.
- [BS04] Jacob Beal and Tim Shepard. Deamplification of dos attacks via puzzles. Draft Copy, October 2004.
- [BS05] Jacob Beal and Tim Shepard. Shrinking the leap of faith. CSAIL Tech Report, March 2005.
- [CA06] Claude Castelluccia and Gildas Avoine. Noisy tags: A pretty good key exchange protocol for RFID tags. In *International Conference on Smart Card Research and Advanced Applications (CARDIS 2006)*, 2006.

- [CG05] Lorrie Faith Cranor and Simson Garfinkel, editors. *Security and Usability: Designing Secure Systems That People Can Use*. O'Reilly & Associates, Inc., Sebastopol, California, 2005.
- [Che] Checkpoint website. Electronic Article Surveillance Vendor. Available from: www.checkpointsystems.com.
- [CJHT] Scott Cantor, John Kemp Jeff Hodges, and Peter Thompson. Liberty id-ff architecture overview. Content as of 8/1/2005.
- [CLL05] E. Y. Choi, S. M Lee, and D. H. Lee. Efficient RFID authentication protocol for ubiquitous computing environment. In *International Workshop on Security in Ubiquitous Computing Systems – SECUBIQ 2005*, Nagasaki, Japan, December 2005.
- [CLRP05] Dario Carluccio, Kerstin Lemke-Rust, and Christof Paar. Electromagnetic side channel analysis of a contactless smart card: First results. In *Workshop on RFID and Light-Weight Crypto*, July 2005.
- [CLRPS06] Dario Carluccio, Kerstin Lemke-Rust, Christof Paar, and Ahmad-Reza Sadeghi. E-passport: The global traceability or how to feel like an UPS package. In Sandra Dominikus, editor, *Workshop on RFID Security 2006 (RFIDSEC '06)*, July 2006.
- [CLRS01] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. The MIT Press, Cambridge, Massachusetts, 2 edition, 2001.
- [Col04] Jonathan Collins. Lost and found in legoland. *RFID Journal*, April 2004. Available from: www.rfidjournal.com/article/articleview/921/1/1/.
- [Col05] Jonathan Collins. Dutch banks follow the money. *RFID Journal*, October 2005. Available from: www.rfidjournal.com/article/articleview/1914/1/4/.
- [Cor] Impinj Corporation. Gen 2 tag clock rage – what you need to know. Impinj RFID Technology Series Whitepaper. Available from: www.impinj.com/files/MR_MZ_TB_00001_TagClockRate.pdf.
- [CvLB06] Christy Chatmon, Tri van Le, and Mike Burmester. Secure anonymous RFID authentication protocols. Technical Report TR-060112, Florida A&M University, Tallahassee, Florida, USA, 2006.

- [DDHV06] Wenliang Du, Jing Deng, Yunghsiang S. Han, and Pramod K. Varshney. A key predistribution scheme for sensor networks using deployment knowledge. *IEEE Transactions on Dependable and Secure Computing*, 3(1):62–77, January–March 2006.
- [Dea05] Identification friend or foe IFF systems: IFF questions & answers, 2005. Available from: www.dean-boys.com/extras/iff/iffqa.html.
- [Dim05] T. Dimitriou. A lightweight RFID protocol to protect against traceability and cloning attacks. In *Conference on Security and Privacy for Emerging Areas in Communication Networks – SecureComm*, Athens, Greece, September 2005. IEEE.
- [Dim06] T. Dimitriou. A secure and efficient RFID protocol that could make big brother (partially) obsolete. In *International Conference on Pervasive Computing and Communications – PerCom 2006*, Pisa, Italy, 2006. IEEE Society Press.
- [Dix05] Douglas Dixon. RFID – tag, you’re it! *Manifest Technology*, November 2005. Available from: www.manifest-tech.com/ce_products/rfid_tags.htm.
- [dJ06] Peter de Jager. Experimenting on humans using alien technology. In Garfinkel and Rosenberg [GR06], chapter 30, pages 439–449.
- [DM02] Inc. Dallas-Maxim. 1-wire SHA-1 overview, September 2002. White Paper 8, APP1201.
- [DMS04] Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: The second-generation onion router. *Proceedings of the 13th USENIX Security Symposium*, pages 303–320, August 2004.
- [DNSa] RFC4033 – DNS security introduction and requirements. RFC4033.
- [dnsb] DNSSEC – DNS security extensions, securing the domain name system. Available from: www.dnssec.net.
- [DO05] Sandra Dominikus and Elisabeth Oswald. Symmetric authentication protocols for RFID systems. *Conference on Pervasive Computing*, 2005.
- [DOI] The doi system. The Digital Object Identifier Webpage. Available from: www.doi.org.
- [Doi05] Yusuke Doi. DNS meets DHT: Treating massive ID resolution using DNS over DHT. *Proceedings of the 2005 Symposium on Applications and the Internet (SAINT ’05)*, 2005.

- [DPLK06] Dang Nguyen Duc, Jaemin Park, Hyunrok Lee, and Kwangjo Kim. Enhancing security of EPCglobal gen-2 RFID tag against tracability and cloning. In *EIECE 2006 Symposium on Cryptography and Information Security*, Hiroshima, Japan, January 2006.
- [DWI⁺05] Yusuke Doi, Shirou Wakayama, Masahiro Ishiyama, Satoshi Ozaki, and Atsushu Inoue. Kallima: A tag-reader protocol for privacy enhanced RFID system. *EUC Workshops*, pages 375–382, 2005.
- [DWIO05] Yusuke Doi, Shirou Wakayama, Masahiro Ishiyama, and Satoshi Ozaki. Privacy enhanced RFID tag with noisy bloom filter. *Conference on Pervasive Computing*, 2005.
- [Dye88] S. P. Dyer. The hesiod name server. In *Proceedings of the USENIX Winter 1988 Technical Conference*, pages 183–190, Berkeley, CA, 1988. USENIX Association. Available from: citeseer.ist.psu.edu/dyer88hesiod.html.
- [E-Z] E-zpass website. Available from: www.ezpass.com.
- [EG02] Laurent Eschenauer and Virgil D. Gligor. A key-management scheme for distributed sensor networks. In *CCS '02: Proceedings of the 9th ACM conference on Computer and communications security*, pages 41–47, New York, NY, USA, 2002. ACM Press.
- [EHDJ04] S. Engberg, M. Harning, and C. Damsgaard-Jensen. Zero-knowledge device authentication: Privacy & security enhanced RFID preserving business value and consumer convenience. In *Conference on Privacy, Security and Trust – PST*, New Brunswick, Canada, October 2004.
- [EPC] EPC Global Website. Available from: www.epcglobalinc.org.
- [EPC05a] EPCglobal. EPC radio-frequency identity protocols class-1 generation-2 UHF RFID: Protocol for communications at 860 mhz – 960 mhz, 2005. Version 1.0.9.
- [EPC05b] The EPCglobal architecture framework, July 2005. Available from: www.epcglobalinc.org/standards/Final-epcglobal-arch20050701.pdf.
- [EPC06] EPCglobal tag data standards version 1.3, March 2006. Available from: www.epcglobalus.org/dnn_epcus/KnowledgeBase/Browse/tabid/277/DMXModule/706/Command/Core_Download/Default.aspx?EntryId=296.

- [FDW04] Martin Feldhofer, Sandra Dominikus, and Johannes Wolkerstorfer. Authentication for RFID systems using the AES algorithm. In *Workshop of Cryptographic Hardware and Embedded Systems - CHES 2004*, volume 3156 of *Lecture Notes in Computer Science*, pages 357–370, Boston, USA, August 2004. Springer.
- [Fel03] Martin Feldhofer. A proposal for authentication protocol in a security layer for RFID smart tags. *Sifting Secure Information and Communication Technologies SIC*, 2003.
- [Fel04] Martin Feldhofer. An authentication protocol in a security layer for RFID smart tags. In *The 12th IEEE Mediterranean Electrotechnical Conference – MELECON 2004*, volume 2, pages 759–762, Dubrovnik, Croatia, May 2004. IEEE.
- [FH01] Simone Fischer-Hubner. *Privacy Enancing Technologies*, volume 1958, chapter 4, pages 107–165. Springer, 2001.
- [FNS75] Horst Fiestel, H. W. Notz, and J. Lynn Smith. Some cryptographic techniques for machine to machine data communications. In *IEEE Proceedings*, volume 64, pages 1545 – 1554, 1975.
- [Fol97] Joe Foley. Glue sniffer: A packet-sniffer detection mechanism, 1997. Developed under MIT’s Network Security Group.
- [Fol99] Joseph Timothy Foley. An infrastructure for electromechanical appliances on the internet. Master’s thesis, Massachusetts Institute of Technology, Cambridge, MA 02139, May 1999.
- [Fol03a] Joseph Timothy Foley. EPC reductionism. *AutoID Center*, November 2003.
- [Fol03b] Joseph Timothy Foley. Klite: A RFID tag anti-counterfeit system. *AutoID Center*, November 2003.
- [FR06] Martin Feldhofer and Christian Rechberger. A case against currently used hash functions in RFID protocols. In Sandra Dominikus, editor, *Workshop on RFID Security 2006 (RFIDSEC ’06)*, July 2006.
- [FS87] A. Fiat and A. Shamir. How to prove yourself: Pracial solutions to identification and signature problems. In *Advances in Cryptology – Crypto ’86*, pages 186–194. Springer-Verlag, 1987.
- [FS03a] Niels Ferguson and Bruce Schneier. *Practical Crytography*. Wiley Publishing, Inc., Indianapolis, Indiana, 2003.

- [FS03b] Niels Ferguson and Bruce Schneier. *Practical Cryptography*, chapter 10, pages 155–184. Volume 1 of Fortuna [FS03a], 2003.
- [FSL04] Christian Floerkemeier, Roland Schneider, and Marc Langheinrich. Scanning with a purpose – supporting the fair information principals in RFID protocols, 2004.
- [FWR05] Martin Feldhofer, Johannes Wolkerstorfer, and Vincent Rijmen. AES implementation on a grain of sand. In *IEEE Proceedings on Information Security*, volume 152, pages 13–20, October 2005.
- [Gar94] Simson Garfinkel. *PGP: Pretty Good Privacy*. O'Reilly & Associates, Inc., 1994.
- [Gar05] Simson Garfinkel. *Design Principles and Patterns for Computer Systems That Are Simultaneously Secure and Usable*. PhD thesis, Massachusetts Institution of Technology, Cambridge, MA 02139, May 2005.
- [GCvD⁺02a] Blaise Gassend, Dwaine Clarke, Marten van Dijk, , and Srinivas Devadas. Controlled physical random functions. In *In Proceedings of the 18th Annual Computer Security Conference*, December 2002. Available from: citeseer.ist.psu.edu/article/gassend02controlled.html.
- [GCvD⁺02b] Blaise Gassend, Dwaine Clarke, Marten van Dijk, , and Srinivas Devadas. Silicon physical random functions, November 2002. Available from: citeseer.csail.mit.edu/gassend02silicon.html.
- [GH06] Simson Garfinkel and Henry Holtzman. Understanding RFID technology. In Garfinkel and Rosenberg [GR06], chapter 2, pages 15–36.
- [GHM⁺03] Martin Gudgin, Marc Hadley, Noah Mendelsohn, Jean-Jacques Moreau, and Henrik Frystyk Nielsen. SOAP version 1.2 part 1: Messaging framework. Technical report, The World Wide Web Consortium, June 2003. Available from: www.w3.org/TR/soap12-part1/.
- [GHT04] Andreas Görlach, Andreas Heinemann, and Wesley W. Terpstra. Survey on location privacy in pervasive computing, 2004. Available from: citeseer.csail.mit.edu/631969.html.
- [Gie] Miek Gieben. DNSSEC history. Available from: www.nlnetlabs.nl/dnssec/history.html.
- [GPJ04] Bourgogne Et Grasset, Paulson, and Bud Jones. RFID gaming chips, October 2004. Available from: www.gpigaming.com/pdf/{RFID}_Gaming_Chips_USA.pdf.

- [GR02] Duncan Graham-Rowe. Exploding chips could foil laptop thieves. *New Scientist*, January 2002. Available from: <http://www.newscientist.com/article.ns?id=dn1795>.
- [GR06] Simson Garfinkel and Beth Rosenberg, editors. *RFID: Applications, Security, and Privacy*. Addison-Wesley, One Lake Street, Upper Saddle River, NJ 97458, 2006.
- [GS1] The GS1 webpage. Available from: www.gs1.org.
- [GSM] GSM cloning. Available from: <http://www.isaac.cs.berkeley.edu/isaac/gsm-faq.html>.
- [GXW⁺04] X. Gao, Z. Xiang, H. Wang, J. Shen, J. Huang, and S. Song. An approach to security and privacy of RFID system for supply chain. In *IEEE International Conference on E-Commerce Technology for Dynamic E-Business*, 2004.
- [Hal94] Neil M. Haller. The S/KEY one-time password system. In *Proceedings of the Symposium on Network and Distributed System Security*, pages 151–157, 1994. Available from: citeseer.ist.psu.edu/haller94skey.html.
- [Hana] Cnri's handle website. www.handle.net.
- [Hanb] The handle system. RFC3650.
- [HB01] Nicholas J. Hopper and Manuel Blum. Secure human identification protocols. In *Advances in Cryptology - ASIACRYPT 2001 : 7th International Conference on the Theory and Application of Cryptology and Information Security*, volume 2248 of *LNCS*, Gold Coast, Australia, December 2001. Springer Berlin / Heidelberg.
- [HM04] D. Henrici and P. Müller. Hash-based enhancement of location privacy for radio-frequency identification devices using varying identifiers. In Ravi Sandhu and Roshan Thomas, editors, *International Workshop on Pervasive Computing and Communication Security - PreSec 2004*, pages 149–153. IEEE Computer Society, March 2004.
- [HmGK05] Gerhard P. Hancke and markus G. Kuhn. An RFID distance bounding protocol. In *Proceedings IEEE/Create-Net SecureComm 2005*, Athens, Greece, September 2005.
- [Hou] House arrest services, inc. Available from: www.housearrest.com.
- [Hou95] The White House. Executive order 12958, April 1995. Available from: www.fas.org/sgp/clinton/eo12958.html.

- [Irv] Irvine sensors website. Available from: www.irvine-sensors.com.
- [isc] Isc internet domain survey. Available from: www.isc.org/index.pl?/ops/ds/.
- [Jam05] Michael S. James. Worth the wait? some resist toll-payment devices. *abc News: Technology and Science*, February 2005. Available from: abcnews.go.com/Technology/Traffic/story?id=491742.
- [JB04] Ari Juels and John Brainard. Soft blocking: Flexible blocker tags on the cheap. In S. De Capitani di Vimercati and P. Syverson, editors, *Workshop on Privacy in the Electronic Society (WPES)*, 2004.
- [Jon06] K.C. Jones. Verichip wants to test human implantable RFID on military. *iTnews Australia*, August 2006. Available from: www.itnews.com.au/newsstory.aspx?CIaID=36214&src=site-marq.
- [JP03] A. Juels and R. Pappu. Squaling euros: Privacy protection in RFID-enabled banknotes. In Rebecca N. Wright, editor, *Financial Cryptography – FC’03*, volume 2742, pages 103 – 121, Guadaloupe, French West Indies, January 2003. Springer-Verlag.
- [JRS03] Ari Juels, Ronald L. Rivest, and Michael Szydlo. The blocker tag: Selective blocking of RFID tags for consumer privacy, 2003. Available from: citeseer.csail.mit.edu/juels03blocker.html.
- [Jue04] Ari Juels. Minimalist cryptography for low-cost RFID tags. In Carlo Blundo and Stelvio Cimato, editors, *International Conference on Security in Communication Networks – SCN 2004*, volume 3352 of *Lecture Notes in Computer Science*, pages 149–164, Amalfi, Italia, September 2004. Springer-Verlag.
- [Jue05a] Ari Juels. Attack on a cryptographic RFID device. *RFID Journal*, February 2005. Available from: <http://www.rfidjournal.com/article/articleview/1415/1/39/>.
- [Jue05b] Ari Juels. Strengthening EPC tags against cloning. In *WiSe ’05: Proceedings of the 4th ACM workshop on Wireless security*, pages 67–76, New York, NY, USA, 2005. ACM Press.
- [JW05] Ari Juels and Stephen Weis. Authenticating pervasive devices with human protocols. In Victor Shoup, editor, *Advances in Cryptology – CRYPTO’05*, volume 3126 of *Lecture Notes in Computer Science*, pages 293–308, Santa Barbara, California, USA, August 2005. IACR, Springer-Verlag.

- [Ker] Generic security service application program interface, version 2, update 1. RFC2743.
- [KFR06] Bernd C. Karl, Stanley M. Finkelstein, and William N. Robiner. The design of an internet-based system to maintain home monitoring adherence by lung transplant recipients. *IEEE Transactions on Information Technology in Biomedicine*, 10(1):66–76, January 2006.
- [KJJ99] Paul Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In M. Wiener, editor, *Advances in Cryptology - CRYPTO'99: 19th Annual International Cryptology Conference*, number 1666 in LNCS, pages 388–397, Santa Barbara, California, August 1999. Springer-Verlag.
- [KN05] Sindhu Karthikeyan and Mikhail Nesterenko. RFID security without extensive cryptography. In *SASN '05: Proceedings of the 3rd ACM workshop on Security of ad hoc and sensor networks*, pages 63–67, New York, NY, USA, 2005. ACM Press.
- [Kob97] Neal Koblitz. Elliptic curve cryptosystems. *Mathematics of Computation*, pages 203–209, 1997.
- [KP06] Sandeep S. Kumar and Christof Paar. Are standards compliant elliptic curve cryptosystems feasible on RFID? In Sandra Dominikus, editor, *Workshop on RFID Security 2006 (RFIDSEC '06)*, pages 41–59, July 2006.
- [KS06] Jonathan Katz and Ji Sun Shin. Parallel and concurrent security of the hb and hb⁺ protocols. In *Eurocrypt 2006*, May 2006.
- [KSCB03] R Koh, E. Schuster, I Chackrabarti, and A Bellman. Securing the pharmaceutical supply chain, 2003. White Paper.
- [LAK06] S. Lee, T. Asana, and K. Kim. RFID mutual authentication scheme based on synchronized secret information. In *Symposium on Cryptography and Information Security*, Hiroshima, Japan, January 2006.
- [Lam81] Leslie Lamport. Password authentication with insecure communication. *Communications of the ACM*, 24(11):770–772, 1981.
- [Lan05a] Jeremy Landt. The history of RFID. *IEEE Potentials*, 24:8 – 11, October–November 2005.
- [Lan05b] Carl E. Landwehr. Green computing. *IEEE Privacy and Security*, 3(6):3, November–December 2005.

- [LCC⁺00] Barry M. Leiner, Vinton G. Cerf, David D. Clark, Robert E. Kahn, Leonard Klienrock, Daniel C. Lynch, Jon Postel, Larry G. Roberts, and Stephen Wolff. A brief history of the internet. *Internet Society Website*, 2000. Snapshot taken on 4/2/2006. Available from: www.isoc.org/internet/history/brief.shtml.
- [lda] What is ldap? Available from: www.gracion.com/server/whatldap.html.
- [LHLL05] S. M. Lee, Y. J. Hwang, D. H. Lee, and J. I. Lim. Efficient authentication for low-cost RFID systems. In Osvaldo Gervasi, Marina Gavrilova, Vipin Kumar, Antonio Laganaà, Heow Pueh Lee, Youngsong Mun, David Daniar, and Chih Jeng Kenneth Tan, editors, *International Conference on Computational Science and its Applications – ICCSA 2005*, volume 3480, pages 619 – 627, Singapore, May 2005. Springer-Verlag.
- [LSMF06] Mikko Lehtonen, Thorsten Staake, Florian Michahelles, and Elgar Fleisch. From identification to authentication – a review of RFID product authentication techniques. In Sandra Dominikus, editor, *Workshop on RFID Security 2006 (RFIDSEC '06)*, pages 91 – 107, July 2006.
- [Man06] Keeping data secure has never been more difficult., January 2006.
- [Mar99] Martin Margala. Low-power sram circuit design. *mtdt*, 00:115, 1999.
- [MD88] P. Mockapetris and K. J. Dunlap. Development of the domain name system. In *ACM SIGCOMM '88: Symposium proceedings on Communications architectures and protocols*, pages 123–133, New York, NY, USA, 1988. ACM Press.
- [Med] Medicalert. Available from: www.medicalert.org.
- [Mil86] Victor S. Miller. Uses of elliptic curves in cryptography. In *Advances in Cryptology – Crypto '85*, pages 417–426. Springer-Verlag, 1986.
- [MNJtH04] Robert Moskowitz, Pekka Nikander, Petri Jokela, and thomas Henderson. Host identity protocol, June 2004. Work in progress. Available from: <http://www.ietf.org/html.charters/hip-charter.html>.
- [MRW04] Josh Mandel, Austin Roach, and Keith Weinstein. Mit proximity card vulnerabilities. MIT CSAIL Presentation given to Stephen D. Immerman, Executive Director of Enterprise Services, John M. McDonald, Associate Director of Enterprise Services, and Daniel L. Michaud, Card Office Manager, March 2004.

- [MSW05] D. Monar, A. Soppera, and D. Wagner. A scalable, delegatable, pseudonym protocol enabling ownership transfer of RFID tags. In *Ecrypt Workshop on RFID and Lightweight Crypto*, July 2005.
- [Mur99] Bill Murray. Army brings dog tags into the electronic age. *Government Computer News*, 18(34), October 1999. Available from: http://www.gcn.com/print/vol18_no34/821-1.html.
- [MW02] Adrian Friday Moamo Wu. Integrating privacy enhancing services in ubiquitous computing environments. *UBICOMP2002*, 2002. Available from: citeseer.csail.mit.edu/wu02integrating.html.
- [MW04] David Molnar and David Wagner. Privacy and security in library RFID: issues, practices, and architectures. In *CCS '04: Proceedings of the 11th ACM conference on Computer and communications security*, pages 210–219, New York, NY, USA, 2004. ACM Press.
- [MZI98] Miodrag Mihaljevic, Yuliang Zheng, and Hideki Imai. A cellular automaton based fast one-way hash function suitable for hardware implementation. *Lecture Notes in Computer Science*, 1431:217–??, 1998. Available from: citeseer.ist.psu.edu/mihaljevic98cellular.html.
- [New06] Annalee Newitz. The RFID hacking underground. *Wired Magazine*, May 2006. Available from: www.wired.com/wired/archive/14.05/rfid.html.
- [nfc] Near field communications group. Available from: www.nfc.org.
- [nis] Protocols.com: Sun protocols family. Available from: www.protocols.com/pbook/sunfamily.htm.
- [NSF06] Z. Nochta, T. Staake, and E. Fleisch. Product specific security features based on RFID technology. In *International Symposium on Application and the Internet Workshops (SAINT-W'06)*, 2006.
- [Nyg99] Erik Nygren. Discussion on ons design with erik nygren <nygren@mit.edu>, 1999.
- [Oas] Oasis uddi. Available from: www.uddi.org.
- [Oas04] Introduction to uddi: Important features and functional concepts, October 2004. Available from: www.oasis-open.org.
- [oD01] US Department of Defense. Protection of sensitive compartmented information, December 2001.

- [p3p] Platform for privacy preferences (P3P) 1.0 (P3P1.0) specification. Available from: www.w3.org/TR/P3P/.
- [Pal96] Chuck Palahniuk. *Fight Club*. Henry Holt and Company, LLC, 1996.
- [PBM05] Andrew S. Patrick, Pamela Briggs, and Stephen Marsh. *Designing Systems That People Will Trust*, chapter 5, pages 75–99. Volume 1 of Cranor and Garfinkel [CG05], 2005.
- [Pea05] J. Pearson. Securing the pharmaceutical supply chain with RFID and public-key infrastructure (pki) technologies., June 2005. White Paper. Available from: www.ti.com/rfid/docs/docntr.shtml.
- [Pec06] Clint Peck. Tracking livestock with RFID. In Garfinkel and Rosenberg [GR06], chapter 15, pages 245–256.
- [PGV93] Bart Preneel, Rene Govaerts, and Joos Vamdewalle. Hash functions based on block ciphers: a synthetic approach. In Douglas R. Stinson, editor, *Advances in Cryptology – CRYPTO’ 93*, number 773 in LNCS, pages 368–378, Santa Barbera, CA, August 1993. Springer-Verlag.
- [Pir06] S. Piramuthu. HB and related lightweight authentication protocols for secure RFID tag/reader authentication. In *Collaborative Electronic Commerce Technology and Research – CUECTeR 2006*, Basel, Switzerland, 2006.
- [PLSP06] Axel Poschmann, Gregor Leander, Kai Schramm, and Christof Paar. DESL: An efficient block cipher for lightweight cryptosystems. In Sandra Dominikus, editor, *Workshop on RFID Security 2006 (RFIDSEC ’06)*, July 2006.
- [Pre03] Loretta W. Prencipe. Battle-tested tech: supply chain and logistics. *InfoWorld*, May 2003. Available from: http://www.infoworld.com/article/03/05/30/22FEbattlesupply_1.html?s=feature.
- [QG90] J.J. Quisquater and L. Guillou. How to explain zero-knowledge protocols to your children. In *Advances in Cryptology – Crypto ’89*, pages 628–631. Springer-Verlag, 1990.
- [RCT05] Melanie Rieback, Bruno Crispo, and Andrew Tanenbaum. RFID guardian: A battery-powered mobile device for RFID privacy management. In Colin Boyd and Juan Manuel González Nieto, editors, *Australasian Conference on Information Security and Privacy – ACISP’05*, volume 3574 of *Lecture Notes in Computer Science*, pages 184–194, Brisbane, Australia, July 2005. Springer-Verlag.

- [RCT06a] Melanic Ricback, Bruno Crispo, and Andrew Tanenbaum. The evolution of RFID security. In *IEEE Pervasive Computing*, volume 5, pages 62–69, January–March 2006.
- [RCT06b] Melanie R. Rieback, Bruno Crispo, and Andrew S. Tanenbaum. Is your cat infected with a computer virus. In *IEEE International Conference on Pervasive Computing and Communications(PerCom)*, Pisa, Italy, March 2006.
- [REC04] Damith C. Ranasinghe, Daniel W. Engels, and Peter H. Cole. Security and privacy: Modest proposals for low-cost RFID systems, September 2004. Available from: www.m-lab.ch/auto-id/SwissReWorkshop/agenda.html.
- [RFI05] The history of RFID technology, October 2005. Available from: www.rfidjournal.com/article/articleprint/1338/1/129/.
- [Riv98] Ron Rivest. Chaffing and winnowing: Confidentiality without encryption. *CryptoBytes(RSA Laboratories)*, 1:12 – 14, Summer 1998. Available from: ftp.rsasecurity.com/pub/cryptobytes/crypto4n1.pdf.
- [RKKW05] K. Rhee, J. Kwak, S. Kim, and D. Won. Challenge-response based RFID authentication protocol for distributed database environment. In Dieter Hutter and markus Ullmann, editors, *International Conference on Security in Pervasive Computing – SPC 2005*, volume 3450, pages 70 – 84, Boppard, Germany, April 2005. Springer-Verlag.
- [RSA06] Jukka Rieki, Timo Salminen, and Ismo Alakärppä. Requesting pervasive services by touching RFID tags. *IEEE Pervasive Computing*, January–March 2006.
- [Rut97] Anthony Rutkowski. Assigning names and numbers: A brief history of the internet domain naming system. *IEEE Internet Computing*, 3(1):10–13, January–February 1997.
- [Sar06] Sanjay Sarma. A history of the EPC. In Garfinkel and Rosenberg [GR06], chapter 3, pages 37–56.
- [SB06] Amdrea Soppera and Trevor Burbridge. Secure by default: The RFID acceptor tag (RAT). In Sandra Dominikus, editor, *Workshop on RFID Security 2006 (RFIDSEC '06)*, July 2006.
- [Sch96] Bruce Schneier. *Applied Cryptography*. John Wiley & Sons, Inc., New York, New York, second edition, 1996.

- [Sec04] RSA Security. What are elliptic curve cryptosystems?, 2004. Available from: www.rsasecurity.com/rsalabs/node.asp?id=2242.
- [Sen] Sensormatic website. Manufacturer of Acousto-Magnetic and RFID Electronic Article Surveillance Systems. Available from: www.sensormatic.com.
- [SF06] Sanjay Sarma and Joe Foley. Discussions about controlling mass tag killing., June 2006.
- [Sha04] Noah Shachtman. Friend or foe? a digital dog tag answers. *New York Times*, April 2004. Available from: www.nytimes.com/2004/04/15/technology/circuits/15next.html?ex=1159243200&en=f98c90cd076e2db4&ei=5070.
- [Sin02] Arundhati Singh. Siren: a sql-based implementation of role-based access control(rbac) for enterprise networks. Master's thesis, Massachusetts Institute of Technology, Cambridge, MA 02139, May 2002.
- [Sta00] Frank Stajano. The resurrecting duckling - what next? In *Security Protocols Workshop*, pages 204–214, 2000.
- [Sta02] Frank Stajano. *Security for Ubiquitous Computing*. John Wiley & Sons, West Sussex, England, 2002.
- [Sta05] Statewide treatment protocols version 6.03-1, November 2005. Available from: www.mass.gov/?pageID=eohhs2terminal&&L=4&L0=Home&L1=Provider&L2=Guidelines+and+Resources&L3=Guidelines+for+Clinical+Treatment&sid=Eeohhs2&b=terminalcontent&f=dph_emergency_services_p_treatment_protocols&csid=Eeohhs2.
- [Sta06] Interagency OPSEC Support Staff. OPSEC glossary of terms, 2006. Available from: www.iooss.gov/docs/definitions.html.
- [Ste08] Robert Louis Stevenson. *The Merry Men; and other Tales and Fables; The Strange Case of Dr. Jekyll and Mr. Hyde (The Complete Works of Robert Louis Stevenson)*. Charles Scribners' Sons, 1908.
- [Ste98] W. Richard Stevens. *Unix Network Programming*, volume 1, appendix A, pages 889–891. Prentice Hall PTR, Upper Saddle River, New Jersey, second edition, 1998.
- [STF05] Thorsten Staake, Frédéric Thiesse, and Elgar Fleisch. Extending the EPC network: the potential of RFID in anti-counterfeiting. In *SAC '05: Proceedings of the 2005 ACM symposium on Applied computing*, pages 1607–1612, New York, NY, USA, 2005. ACM Press.

- [Sto48] Harry Stockman. Communication by means of reflected power. *Proceedings of the I.R.E.*, 36:1196–1204, 1948.
- [Sul04] Laurie Sullivan. Cattle trails: RFID technology could be used to build a national livestock-tracking system. *Information Week*, January 2004. Available from: <http://www.informationweek.com/story/showArticle.jhtml?articleID=17300330>.
- [Tan03] Andrew S. Tanenbaum. *Computer Networks*. Prentice Hall PTR, 4 edition, 2003.
- [TB06] Pim Tuyls and Lejla Batina. RFID-tags for anti-counterfeiting. In David Pointcheval, editor, *Topics in Cryptology - CT-RSA 2006, The Cryptographers' Track at the RSA Conference 2006*, Lecture Notes in Computer Science, San Jose, CA, USA, February 2006. Springer-Verlag.
- [Teca] PTA Technologies. Practical threat analysis for securing computerized systems. Available from: www.ptatechnologies.com/Documents/PTA_for_Software.pdf.
- [Tech] PTA Technologies. Pta(practical threat analysis) methodology in a nutshell. Available from: www.ptatechnologies.com/Documents/PTA_InANutshell.pdf.
- [TLS] The transport layer security (TLS) protocol version 1.1. RFC4346.
- [Tor06] The onion router(Tor) webpage, 2006. tor.eff.org.
- [Tre05] Bob Trebilcock. Adding RFID to pallets. *Modern Materials Handling*, February 2005. Available from: www.mmh.com/article/CA503820.html.
- [Tsu06] G. Tsudik. YA-TRAP: Yet another trivial RFID authentication protocol. In *International Conference on Pervasive Computing and Communications – PerCOM 2006*, Pisa, Italy, March 2006. IEEE Computer Society Press.
- [Uhl96] Jim Uhls. Fight club screenplay, 1996. Based upon a novel by Chuck Palahniuk.
- [VB03] Istvan Vajda and Levente Buttyan. Lightweight authentication protocols for low-cost RFID tags. *Ubiquitous Computing(UBICOMP)*, 2003. Available from: citeseer.ist.psu.edu/vajda03lightweight.html.
- [Ver] VeriChip website. Available from: www.verichipcorp.com.

- [WCO00] Larry Wall, Tom Christiansen, and Jon Orwant. *Programming Perl*. O'Reilly & Associates, Inc., Sebastopol, California, third edition, 2000.
- [WE] Jim Waldrop and Dan Engles. Savant 1.0.
- [Wei03] Stephen August Weis. Security and privacy in radio-frequency identification devices. Master's thesis, Massachusetts Institute of Technology, Cambridge, MA 02139, May 2003.
- [Wei04] Will Weissert. Microchips implanted in mexican officials. *MSNBC Associated Press*, July 2004. Available from: www.msnbc.msn.com/id/5439055/.
- [Wel] Brian Wellington. dnsjava – DNS written in java. Available from: www.dnsjava.org.
- [Wes01] Dr. Alan Westin. Opinion surveys: What consumers have to say about information privacy. Technical report, The House Committee on Energy and Commerce, May 2001. Available from: <http://energycommerce.house.gov/107/hearings/05082001Hearing209/Westin309.htm>.
- [Wes06] Jonathan Westhues. Hacking the prox card. In Garfinkel and Rosenberg [GR06], chapter 19, pages 291–301.
- [WSRE03] Steven Weis, Sanjay Sarma, Ron Rivest, and Dan Engles. Security and privacy aspects of low-cost radio frequency identification systems. In *International Conference on Security in Pervasive Computing – SPC 2003*, volume 2802, pages 454 – 469. Springer-Verlag, 2003.
- [XML] XML-RPC Website. Available from: www.xmlrpc.com.
- [YPL⁺05] J. Yang, J. Park, H. Lee, K. Ren, and K. Kim. Mutual authentication protocol for low-cost RFID. In *Ecrypt Workshop on RFID and Lightweight Crypto*, July 2005.
- [YSH06] Pengyuan Yu, Patrick Schaumont, and Dong Ha. Securing RFID with ultra-wideband modulation. In Sandra Dominikus, editor, *Workshop on RFID Security 2006 (RFIDSEC '06)*, July 2006.
- [YY03] Kim Young-Young. Radio ID chips may track banknotes. *CNet News.com*, May 2003. Available from: news.com.com/2100-1017-1009155.html.
- [Zim94] Philip Zimmermann. Pretty good(tm) privacy, public key encryption for the masses: User guide. Version 2.6.2, October 1994.

- [ZK05] X. Zhang and B. King. Integrity improvements to an RFID privacy protections protocol for anti-counterfeiting. In Jianying Zhou, Javier Lopez, Robert Deng, and Feng Bao, editors, *Information Security Conference – ISC 2005*, volume 3650, pages 474–481, Singapore, September 2005. Springer-Verlag.