SANDIA REPORT

SAND2007-5991
Unlimited Release
Printed September 2007

Low-Bandwidth Authentication

Michael J. Collins, Erik Anderson, Lauren McIver, Patrick Donnelley, Brian Gaines, Austin McDaniel, Kurt Thomas

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550

Sandia is a multiprogram laboratory operated by Sandia Corporation,
a Lockheed Martin Company, for the United States Department of Energy's
National Nuclear Security Administration under Contract DE-AC04-94-AL85000.

Sandia National Laboratories

SAND2007-5991
Unlimited Release
Printed September 2007

Low-Bandwidth Authentication

Michael J. Collins, Erik Anderson, Lauren McIver
Cryptography and Information Surety Dept. 5614
Sandia National Laboratories
P.O. Box 5800
Albuquerque, NM 87185-0672
mjcolli@sandia.gov
weander@sandia.gov

Patrick Donnelley, Brian Gaines
Austin McDaniel, Kurt Thomas
Networked Systems Dept. 5616
Sandia National Laboratories
P.O. Box 5800
Albuquerque, NM 87185-1372

Abstract

Remotely-fielded unattended sensor networks generally must operate at very low power — in the milliwatt or microwatt range — and thus have extremely limited communications bandwidth. Such sensors might be asleep most of the time to conserve power, waking only occasionally to transmit a few bits. RFID tags for tracking or material control have similarly tight bandwidth constraints, and emerging nanotechnology devices will be even more limited. Since transmitted data is subject to spoofing, and since sensors might be located in uncontrolled environments vulnerable to physical tampering, the high-consequence data generated by such systems must be protected by cryptographically sound authentication mechanisms; but such mechanisms are often lacking in current sensor networks. One reason for this undesirable situation is that standard authentication

methods become impractical or impossible when bandwidth is severely constrained; if messages are small, a standard digital signature or HMAC will be many times larger than the message itself, yet it might be possible to spare only a few extra bits per message for security. Furthermore, the authentication tags themselves are only one part of cryptographic overhead, as key management functions (distributing, changing, and revoking keys) consume still more bandwidth.

To address this problem, we have developed algorithms that provide secure authentication while adding very little communication overhead. Such techniques will make it possible to add strong cryptographic guarantees of data integrity to a much wider range of systems.

Acknowledgment

Many thanks to Karen Shanklin for recruiting an outstanding group of summer interns to participate in this project.

Many thanks to Carl Diegert and Scott Mitchell for helpful technical discussions.

Many thanks to Rolf Riesen for developing and maintaining the LaTex SAND report class which was used to typeset this report.

Contents

1 Introduction **11**

2 Independent-Subset Authentication **13**

2.1 Introduction 13

2.2 Subset Authentication 13

2.2.1 Sliding-Window Construction 15

2.2.2 Optimal Sequences for Small Memory Bounds 18

2.2.3 Lower Bounds for the Sliding Window Construction 18

2.2.4 Iterative Improvement of Windows 19

2.3 More General Independence Conditions 19

3 Bloom-Filter Authentication **21**

3.1 Bloom Filters 21

3.2 Authentication Using Bloom Filters 23

3.2.1 Preventing Attacks against Bloom Filters 24

3.3 Compressing Bloom Filters 24

4 Hybrid Authentication **27**

4.1 Overview 27

4.2 Stand-Alone Block Authentication 27

4.2.1 Application Data Transmission 27

4.2.2 Authentication Data Transmission 28

4.2.3 Stand-Alone Communication Outline 28

4.2.4 Performance 29

4.3 Segmented Authentication 30

4.3.1 Segmented Communication Outline 30

4.4 Throttling Authentication 31

5 Signal-Strength Authentication **33**

5.1 Introduction 33

5.2 The Circuit 34

5.3 Broadcasting High Power Messages 35

5.4 Countering Specific Attacks 35

5.4.1 Worm Hole 35

5.4.2 HELLO floods 36

5.4.3 Sinkholes 36

5.5 Unresolved Concerns 36

References **37**

List of Figures

3.1 Bloom filter Insertion and Query 22

3.2 Varying Collision Probability Based on m:n 23

5.1 Circuit Diagram. 34

List of Tables

2.1 Optimal $q(b)$ for small v 18

2.2 Best known $q(b)$ for large v 19

2.3 Best known $q^{r'}$ for various v, r' 20

2.4 Best known $q_2(b)$ for some v 20

3.1 Maintaining a constant compression size z. 25

3.2 Maintaining a constant false-positive rate. 25

Chapter 1

Introduction

We consider a variety of technical approaches to the problem of secure communication for severely resource-constrained wireless sensor networks. In chapter 2 we consider the problem of minimizing the number of bits which must be appended to each message in a long stream in order to attain a desired level of security on the stream. We formulate this as a discrete optimization problem on subsets of the message stream and prove several results. In chapter 3 we consider another method for attaining the same goal, using Bloom filters [2] in a novel way to compress authentication bits. Chapter 4 considers yet another method, which can be used if the network allows two-way communication between a sensor node and its base station. Finally, chapter 5 considers how to protect low-power nodes against resource-exhaustion attacks by detecting malicious messages in hardware using a simple circuit to analyze signal strength.

Chapter 2

Independent-Subset Authentication

2.1 Introduction

We wish to send a stream of many short messages $m_1, m_2, m_3, \cdots$ on a channel with very limited bandwidth, and we wish to provide strong cryptographic authentication for this data. Because bandwidth is so limited, we assume that we must use almost all transmitted bits for delivering payload data: say we can append no more than r bits of authentication to each message, where r is too small to provide adequate security. Suppose we have decided that qr authentication bits are needed for security; a simple solution would be to send q consecutive messages $m_1, m_2, \cdots m_q$, followed by a message authentication tag t of length qr for the concatenated message $(m_1|m_2|\cdots m_q)$ (repeating this process for the next block of q messages and so on). This achieves the desired data rate, but it is unsatisfactory for several reasons. In an extremely low-power environment (such as a wireless network of very small sensors), we expect that many messages will be dropped or corrupted, making it impossible for the receiver to verify the correctness of t. Also, we are transmitting no data at all during the relatively long time needed to transmit the tag. We seek a more robust solution which will tolerate some missing messages (without the additional cost of applying an error-correction scheme to already-redundant data), and which does not interrupt the flow of payload data.

2.2 Subset Authentication

Our basic approach is to append a short authentication tag a_i to each message m_i; each a_i is an r-bit authentication tag for some appropriately-chosen subset S_i of the previous messages. Let $\mathscr{A}_K$ be a message authentication code (MAC) with key K that produces an r-bit output. Thus if $S_i = \{j_1^i < j_2^i < \cdots j_k^i\}$ we have[1]

$$a_i = \mathscr{A}_K(i|m_{j_1^i}|\cdots|m_{j_k^i}) \tag{2.2.1}$$

and we transmit $m_1, a_1, m_2, a_2, \cdots$. If each message is contained in q sets, then each message is used in the computation of q different tags, and we will eventually accumulate

[1] It is convenient to ignore the distinction between a message and its index, writing $j \in S_i$ instead of $m_j \in S_i$.

the required qr bits of authentication for each message. If $\mathscr{A}_K$ is a pseudorandom function, an adversary cannot cause an invalid m_i to be accepted without guessing qr random bits. In practice, $\mathscr{A}_K$ could be implemented by truncating the output of a full-length authentication code such as HMAC [1]. The design of a secure, less computationally intensive MAC which inherently produces a short output would pose an interesting and challenging problem.

However, it is not enough to simply require that each message appear q times. We are assuming a very low-power network with no acknowledgement or retransmission protocol, no error-correction mechanism, and occasional loss of connectivity. Thus we must expect that some messages will be lost, and if m_j is lost, all tags a_i such that $j \in S_i$ will be useless. Therefore every message must be contained in more than q sets, to provide robustness against the expected missing messages. The question is, what conditions must we impose on the sets S_i, and what is the optimal way to achieve those conditions?

We first consider the following requirement (more general requirements are considered in section 2.3): if any one message is lost, this must not prevent full authentication of any other message. This means that for any pair of messages m_i, m_j, we must have at least q sets which contain m_i but not m_j. Thus if m_j is lost, we still have enough good tags to authenticate m_i with the desired degree of security.

This "set-cover" approach requires the sender to remember many old messages. If a node can remember at most v old messages, then we must have $S_i \subset [i-v, i]$ for all i. Memory is presumably quite limited since we are dealing with very low-power nodes. Note that v is also the maximum delay before a message finally achieves full authentication, which is another reason to limit v.

Thus we have the following problem: Given memory bound v, find sets S_i that maximize q where

- For each $i \in \mathbb{N}$, $S_i \subset [i-v, i]$
- For each $i \neq j$ there are at least q sets S with $i \in S$, $j \notin S$

Given a collection of sets S, define the *strength* of the collection as

Definition 2.2.1.

$$q(S) = \min_{i,j} \#\{t | i \in S_t, j \notin S_t\}$$

(here $\#A$ denotes the size of a set A). We have defined S as an infinite collection; such a collection would of course be specified either by rotating through a finite collection of given sets, or by specifying a way to generate S_i as a function of S_{i-1}. To get the process started, we can implicitly have dummy messages $m_{-v}, \cdots m_{-1}, m_0 = 0$.

2.2.1 Sliding-Window Construction

We first consider the special case in which each set S_i is defined by a "sliding window"; we select a set of distances $\delta = \{\delta_1 < \cdots \delta_k \leq v\}$ and let each $S_i = \{i - \delta_1, \cdots i - \delta_k\}$.

It will be convenient to identify the vector of distances d with a binary sequence b of length v which is zero except on δ. Then

$$S_i = \{i - d : b_d = 1\}.$$

We may also treat b as an infinite sequence with $b_j = 0$ for j outside of the interval $[0, v-1]$. We say that difference d is "realized (at j)" if $b_j = 1, b_{j+d} = 0$ and call the ordered pair $(j, j+d)$ a "realization of d". We define

Definition 2.2.2.

$$\mathrm{rel}_b(d) = \#\{i | b_i = 1, b_{i+d} = 0\} \tag{2.2.2}$$

so $\mathrm{rel}_b(d)$ is the number of times d is realized in b (we may drop the subscript b when the context is clear). We then define the strength of the vector b as

$$q(b) = \min_d \mathrm{rel}_b(d) \tag{2.2.3}$$

consistent with the definition given above for $q(S)$.

We can assume with no loss of generality that $b_0 = b_{v-1} = 1$. Changing b_0 from zero to one does not destroy any realizations of any d; changing b_{v-1} from zero to one creates one new realization of d for every d, while destroying one realization of each d with $b_{v-d-1} = 1$. Thus $q(b)$ might increase and cannot decrease.

Note that we do not need to consider differences with absolute value greater than v; for such differences we clearly have $\mathrm{rel}(d) = \sum_i b_i$, which is a trivial upper bound on all $\mathrm{rel}(d)$. In fact we can limit our attention to positive differences:

Lemma 2.2.1. *For all d, rel(d) = rel$(-d)$*

Proof. $\mathrm{rel}(d) - \mathrm{rel}(-d) = \sum_i (b_i - b_{i+d}) = 0$ □

The problem of maximizing $q(b)$ for a fixed v has apparent connections to several other problems, in particular to the problems of optimal autocorrelation and side-lobe minimization; we have

$$\mathrm{rel}(d) = \sum_i b_i(1 - b_{i+d}) = k - \sum_i b_i b_{i+d} \tag{2.2.4}$$

Letting $\hat{b}^d$ be the sequence $0^d b 0^d$ and assuming $d < k$, and denoting the aperiodic autocorrelation [4] of a finite binary sequence s by $AA(s)$ we have

Theorem 2.2.2. *For all d, rel(d) = $\frac{v+d-AA_d(\hat{b}^d)}{4}$*

Proof. By lemma 2.2.1, we have that $\text{rel}(d) = \frac{1}{2}(\text{rel}(d) + \text{rel}(-d))$, which is one-half the number of pairs $(i, i+d)$ with $b_i \neq b_{i+d}$. And $AA_d(\hat{b}^d)$ is the number of such pairs with $b_i \neq b_{i+d}$ minus the number of pairs with $b_i = b_{i+d}$. There are $v + d$ such pairs overall, thus

$$AA_d(\hat{b}^d) = v + d - 4\text{rel}(d) \tag{2.2.5}$$

□

We can bound the maximum strength of a sequence for a given memory size v as follows:

Theorem 2.2.3. *For all b of length v,*

$$q(b) \leq \frac{v+2}{3} \tag{2.2.6}$$

Proof. We in fact prove the stronger result that

$$\min(\text{rel}(1), \text{rel}(2)) \leq \frac{v+2}{3} \tag{2.2.7}$$

Let R^s_ℓ be the number of runs of $s \in \{0, 1\}$ of length ℓ. With no loss of generality we may assume that $\ell \leq 2$; in a long run of ones or zeros, the third value can be changed without decreasing rel(1) or rel(2). Then we have

$$v = R^0_1 + R^1_1 + 2R^0_2 + 2R^1_2 \tag{2.2.8}$$

Runs of zeros and ones alternate, and we can assume with no loss of generality that the sequence starts and ends with 1, so we also have

$$R^1_1 + R^1_2 = 1 + R^0_1 + R^0_2 \tag{2.2.9}$$

and combining these we obtain

$$v = 2(R^1_1 + R^1_2) + R^0_2 + R^1_2 - 1 \tag{2.2.10}$$

Now $\text{rel}(1) = R^1_1 + R^1_2$ since this is the number of runs of ones. Furthermore, the distance 2 will fail to be realized at $b_i = 1$ if and only if this is immediately followed by a zero-run of length one; thus (using equation 2.2.9)

$$\text{rel}(2) = R^1_1 + 2R^1_2 - R^0_1 = 1 + R^0_2 + R^1_2 \tag{2.2.11}$$

therefore

$$v = 2\text{rel}(1) + \text{rel}(2) - 2 \tag{2.2.12}$$

and the theorem follows. □

In fact, the same bound applies to any collection of sets, without the sliding-window assumption:

Theorem 2.2.4. *For any collection of sets S with memory bound v,*

$$q(S) \leq \frac{v+2}{3} \tag{2.2.13}$$

Proof. Let b^i be the binary sequence corresponding to the set S_i, i.e. $b^i_t = 1$ if and only if $i - t \in S_i$. Consider v consecutive sets $S_i, \cdots S_{i+v-1}$. These are the only sets which can contain i; thus for any distance d, at least $q(S)$ of these sets contain i but not $i+d$. Thus the sequences $b^i \cdots b^{i+v-1}$ together contain at least $q(S)$ realizations of d, where in sequence b^{i+t} we only count a realization at bit position t.

Similarly, the sequences $b^{i+1} \cdots b^{i+v}$ contain at least $q(S)$ *different* realizations of d and so, for any L, the $v+L-1$ sequences $b^i \cdots b^{i+v+L-2}$ contain qL different realizations of d. Thus as L approaches infinity, the average value of $\mathrm{rel}_{b^j}(d)$ for $i \leq j \leq i+v+L-2$ approaches (at least) $q(S)$. In particular this holds for $d = 1, 2$. Now from the proof of theorem 2.2.3, we know that

$$2\mathrm{rel}_{b^j}(1) + \mathrm{rel}_{b^j}(2) \leq v + 2$$

for each sequence b^j, thus the same must be true of the averages, i.e.

$$3q(S) \leq v + 2$$

□

It is not known whether the strength of an arbitrary collection of sets can exceed the maximum strength achievable by a sliding window. The proof of theorem 2.2.4 shows that if this is the case, we must have a collection of sliding windows in which the average value of each $\mathrm{rel}(d)$ exceeds the maximum strength of any single sliding window.

We also have the following relationship among different distances:

Theorem 2.2.5. *For all d, d'*

$$rel(d) + rel(d') \geq rel(d + d') \tag{2.2.14}$$

In particular,

$$2rel(d) \geq rel(2d)$$

Proof. If $d \neq d'$ define a mapping from realizations of $d + d'$ to realizations of d and d' as follows: for each $b_i > b_{i+d+d'}$, map $(i, i+d+d')$ to $(i, i+d)$ if $b_{i+d} = 0$, else map to $(i+d, i+d+d')$. Clearly this map is injective.

If $d = d'$ then similarly every realization of $2d$ can be mapped to exactly one realization of d, and no more than two realizations of $2d$ can map to the same point. □

v	$\max q(v)$	an optimal vector
1	1	1
4	2	1 1 0 1
7	3	1 1 0 0 1 0 1
10	4	1 1 0 1 0 1 0 0 1 1
14	5	1 1 1 0 0 1 0 1 0 1 1 0 0 1
17	6	1 1 1 0 0 1 0 1 1 0 0 1 1 0 1 0 1
21	7	1 1 1 0 0 0 1 0 1 0 1 1 0 1 0 0 1 1 0 0 1
24	8	1 1 1 0 0 0 1 0 1 1 0 1 0 0 1 1 0 0 1 1 0 1 0 1
27	9	1 1 1 0 0 1 0 1 0 1 0 1 1 0 0 1 0 1 1 0 0 0 1 1 0 1 1
31	10	1 1 1 1 0 0 0 1 1 0 1 0 1 0 0 1 1 0 0 1 1 0 1 0 0 1 1 0 1 0 1
35	11	1 1 0 1 0 1 0 0 1 0 0 1 1 1 1 0 0 1 1 0 0 0 1 0 1 1 0 0 1 0 1 0 1 1 1

Table 2.1. Optimal $q(b)$ for small v

2.2.2 Optimal Sequences for Small Memory Bounds

For small values of v, optimal sequences can be found by exhaustive search; results are summarized in table 2.1. Only "critical" values are shown, i.e. v at which the maximum $q(b)$ changes. These results show that the bound of theorem 2.2.3 can be attained for small v. For all lengths except $v = 35$, the table gives the lexicographically smallest vector attaining $\max q(b)$. Exhaustive search was not completed for $v = 35$, but $q(b) = 11$ is still known to be optimal; if we had b of length 35 attaining $q(b) = 12$, we could remove one bit to obtain $q(b) = 11$ at length 34, which has been ruled out by exhaustive search.

As a secondary objective, we could seek to minimize the Hamming weight of b: this weight is the number of messages that must be combined to compute each authentication tag, so reducing this weight may reduce the amount of work needed to compute a_i. For all v in this table, the majority of optimal vectors have weight greater than $\frac{v}{2}$; for all these v (except 21 and possibly 35) there are no optimal vectors with weight less than $\frac{v}{2}$.

2.2.3 Lower Bounds for the Sliding Window Construction

If $v + 1$ is a power of 2 we can attain $q(b) \geq \frac{v+1}{4}$ by letting b be the output of a linear feedback shift register [5] with period v. The perfect autocorrelation of an LFSR implies that $\frac{v+1}{2}$ bit positions will have $b_i = 1$, and for any $d > 0$ exactly half of these will have $b_{i+d} = 0$ where the indices are taken modulo v.

More generally, if a difference set [6] D of size v exists, then we attain $q(b) \geq \frac{v+1}{4}$ by letting $b_i = 1$ precisely when $i \in D$. A difference set is a set of $k = \frac{v-1}{2}$ integers mod v, such that for each $d > 0$, there are exactly $\frac{k-1}{2}$ pairs $a, b \in D$ with $a - b = d$; this implies that there are exactly $\frac{k+1}{2}$ pairs $a \in D, b \notin D$ with $a - b = d$, hence $\text{rel}(d) \geq \frac{k+1}{2} = \frac{v+1}{4}$.

v	max known $q(b)$	Min weight attaining max $q(b)$
40	12	19
60	18	30
100	28	48
200	55	100
300	81	150

Table 2.2. Best known $q(b)$ for large v

LFSRs and difference sets are, however, periodic structures which do not take advantage of the edge effects inherent in this problem, and they do not provide optimal solutions. It appears to be possible to do somewhat better than $\frac{v+1}{4}$ for all v (see section 2.2.4), although it also seems that the maximum $q(b)/v$ approaches $\frac{1}{4}$ as v approaches infinity.

2.2.4 Iterative Improvement of Windows

Starting with a random binary vector b^0, we can attempt to maximize q by iterative local improvement. At each step, we change one bit of the current solution b^i. If we can attain $q(b^{i+1}) > q(b^i)$ by flipping a single bit, we do this (note that a single bit change cannot increase the strength of the vector by more than 1, since it cannot change any $\text{rel}(d)$ by more than 1). If such immediate improvement is not possible, we consider the set of distances d which are tight, i.e. which have $\text{rel}(d) = q(b^i)$. The local optimization criteria is to reduce the size of this set as much as possible, subject to the condition that strength does not decrease (i.e. that there is no d for which $\text{rel}(d)$ decreases to $q(b^i) - 1$). If local improvement is impossible, we flip two bits at random.

In order to implement this search, note that it is not necessary to recompute q from scratch for every vector at Hamming distance 1 from b^i. Instead, for each bit position i and for each tight distance d, we can compute in constant time the effect on $\text{rel}(d)$ of flipping bit i. Table 2.2 gives the strengths of the best vectors found in this manner.

2.3 More General Independence Conditions

More generally we may consider conditions of the following form: for parameters (r, r'), require that loss of any r messages does not prevent authentication of more than r' remaining messages. The problem considered above is the special case $r = 1, r' = 0$. In the general case we have the following: for any set A with $\#A = r$, there can be no more than r' indices $i \notin A$ such that

$$\#\{j | i \in S_j, A \cap S_j = \emptyset\} < q$$

This is a difficult condition to deal with in general, so we still consider some special cases, and still consider only the sliding-window approach. If we have $r = 1$ but $r' > 0$, then we no are no longer maximizing the minimum value of $\mathrm{rel}(d)$; instead we seek to maximize the $(r' + 1)^{\text{th}}$ smallest value. The r' smallest values correspond to the r' messages for which we are allowed to loose full authentication when just one message is altered. Given b we denote the $(r'+1)^{\text{th}}$ smallest value of $\mathrm{rel}(d)$ by $q^{r'}(b)$. Table 2.3 gives the best known results (obtained by iterative search) for some small values of v and r'.

$r' \backslash v$	10	20	30	40	60	100
1	4	7	10	13	19	30
2	5	8	11	14	20	30
4	5	10	13	16	22	32
6		10	14	17	24	34
8		10	15	19	25	36
10			15	20	26	38
15				20	30	43
20					30	46

Table 2.3. Best known $q^{r'}$ for various v, r'

If we have $r > 1$ and $r' = 0$, then we require that the loss of any set of r messages does not prevent authentication of any other message. For this case we define $\mathrm{rel}(d_1, d_2, \cdots d_r)$ as the number of indices j where $b_j = 1, b_{j+d_1} = \cdots = b_{j+d_r} = 0$, and maximize $q_r(b) = \min \mathrm{rel}(d)$ over all vectors d, where we may assume $i < j$ implies $d_i < d_j$ since order does not matter. Note that the d_i may be negative. Trivially we have

$$q_r(b) \leq \frac{v + r}{r + 1} \tag{2.3.1}$$

since every realization of $d = (1, 2, \cdots r)$ (except at $b_{v-1} = 1$) consists of a 1 followed by r zeros, and these cannot overlap. Table 2.4 gives the best known values of q_2 and q_3 (again obtained by iterative search) for various memory bounds v.

v	max known $q_2(b)$	max known $q_3(b)$
10	2	1
20	4	3
30	5	3
40	7	4
60	10	5
100	16	8

Table 2.4. Best known $q_2(b)$ for some v

Chapter 3

Bloom-Filter Authentication

3.1 Bloom Filters

The independent-subset approach to low-bandwidth authentication might become impractical if we need to protect against a large number of dropped messages. Here we consider an alternative approach which utilizes Bloom filters to authenticate a set of messages. Bloom filters allow for each individual message to be verified independent of the other messages being received or lost. The loss of any set of messages will not prevent the authentication of any other message. This feature allows Bloom filters to scale well in systems where noise and data corruption are highly probable. The disadvantage of the Bloom filter approach is that it requires more authentication bits per message.

A Bloom filter is a compressed data structure for storing a set S, supporting queries of the type "is x in S". Bloom filters guarantee searches will produce no false-negatives, but there can be false-positives; i.e. there is a small chance that a query will produce the answer "yes" even though $x \notin S$. Items can be added but not removed, and the existence of an item can be tested in constant time.

The classic Bloom filter, as introduced in [2], is initialized as an m-bit array with each element set to zero. To add an item x into the filter, the item is processed by k hash functions derived from two distinct hash functions H_a, H_b in such a way that each of the k hash functions are unique:

$$H_i(x) = H_a(x) + i \times H_b(x) (\bmod m)$$

Each of these hash functions will map to a single bit of the m-bit array. This array cell is then set to 1. Once an array cell is set to 1 it will never change back to zero: if a second item maps to a cell which is already set to 1, nothing will change.

To check whether an item is stored in the Bloom filter, the same process of hashing the element k times is repeated. The bit positions in the array are inspected. If any of the bit mappings point to a cell that is still set to zero then there is absolute certainty that the item is not in the Bloom filter. Should all of the bits reference cells set to one, we can conclude with some probability that the item is within the Bloom filter.

void AddToBloom(message)

```
for (i = 1 ... k) {
mapping = H(i,message);
Bloom(mapping) = 1;
}
```

boolean CheckInBloom(message)

```
for (i = 1 ... k) {
check = H(i)(message);
if (Bloom(check) ≠ 1) {
quit(failure: message is not in the Bloom filter);
}
}
quit(success: all bits of the message match those of the Bloom filter);
```

Figure 3.1. Bloom filter Insertion and Query

For a Bloom filter that is m-bits long, constructed from k hash functions, and storing n messages, the probability for a false positive is the probability that k randomly-chosen bits have been set to 1:

$$\left(1-\left(1-\frac{1}{m}\right)^{kn}\right)^{k} \approx \left(1-e^{-kn/m}\right)^{k} \tag{3.1.1}$$

If m and n are known, then k can be optimized to minimize the probability of false positives. This probability is minimized when each cell has a $\frac{1}{2}$ probability of equaling one, thus

$$k = \frac{m}{n} ln(2) \tag{3.1.2}$$

and the probability of a false positive is

$$\left(\frac{1}{2}\right)^{k} \approx 0.6185^{m/n} \tag{3.1.3}$$

As m increases for fixed n, the probability of a collision decreases logarithmically.

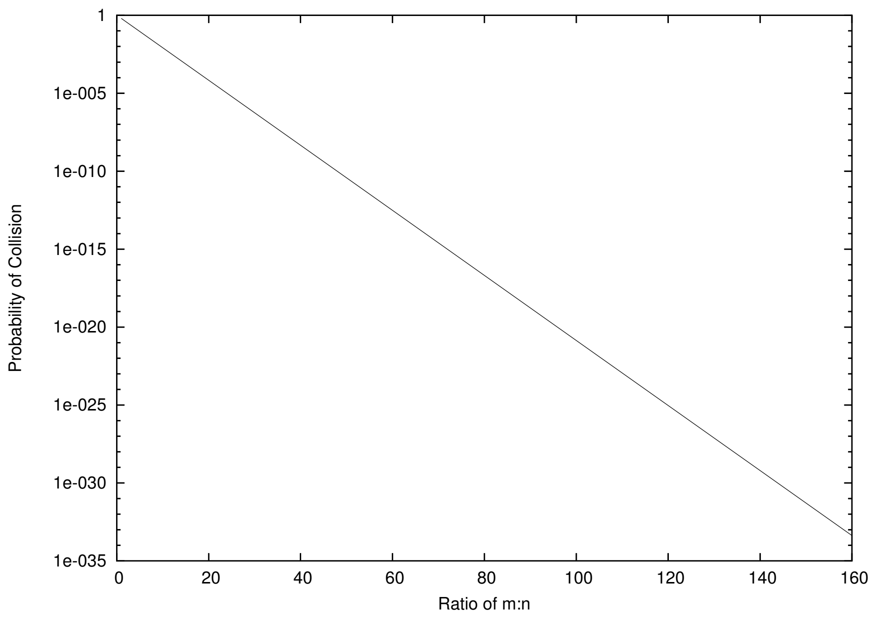

Figure 3.2. Varying Collision Probability Based on m:n

3.2 Authentication Using Bloom Filters

In order to use Bloom filters for authentication, we use *keyed* hashes H_a^K, H_b^K. The basic idea is that after sending n messages, we authenticate them by sending the keyed Bloom filter containing these messages. The receiving node authenticates each message m by checking if m is in the filter. An adversary who does not know K cannot compute the correct filter; the probability that a forged m will be accepted is the probability of a false positive in the filter.

Actually, in the low-bandwidth setting, we do not want to send the entire filter all at once. Instead the sender $Node_A$ begins by gathering a set of n messages and sending each of these n values to receiver $Node_B$ with no authentication provided. Once a total of n messages have been sent by $Node_A$, the next round of sending a new set of n messages begins and authentication data for the previous n messages is appended to each outgoing packet. To each message in the next set of n messages, we will append m/n bits of the Bloom filter for the last set of n messages.

For each message r received, $Node_B$ will test inclusion within the Bloom filter, checking to see that each of the k keyed hash functions reference cells that are set to 1. Should any of the Bloom filter bits be lost during transmission, mappings into those bits will be ignored for authentication. Thus for an m-bit array with n total sections, with $s \times n$

sections lost where $0 \leq s < 1$, the remaining $n - s \times n$ sections must provide adequate security. For each message we expect that $(1 - s)k$ of the hash functions will map into the portion of the array received by $Node_B$, and the probability of a false positive is

$$\left(\frac{1}{2}\right)^{(1-s)k} \tag{3.2.1}$$

Once the set of messages has been authenticated the Bloom filter can be removed from memory or overwritten by the Bloom filter for the next block of messages.

3.2.1 Preventing Attacks against Bloom Filters

A simple attack on this scheme would be for an adversary to transmit a Bloom filter in which all bits are set to 1, so that all messages will authenticate. This could be prevented by encrypting the filter, or by putting an upper bound on the fraction of bits set to 1. The latter option is preferable since it does not require an extra encryption/decryption operation at each stage. Given the parameters of the filter we know how many bits we expect to set to 1

3.3 Compressing Bloom Filters

Application of Bloom filters to low-bandwidth authentication requires a very low false-positive rate, but also requires a small filter, and these requirements conflict. The compressed Bloom filter, first described in [7], makes a tradeoff, reducing the efficiency of storing items in order to allow for compression of the filter. For the optimized case above, the probability of any single bit of the m-bit array being set to one is $\frac{1}{2}$. Since the hash functions are essentially random, it is impossible to compress such a filter. To enable compression, the probability that any array bit is set to one is reduced to less than $\frac{1}{2}$. This can be accomplished by either increasing the size of m or reducing the number of k, in either case, effecting the size of the Bloom filter or the original false positive rate. The final result is a compressed size $z < m$.

The savings possible for compressing a Bloom filter to z bits is dependent on the entropy of the binary array. The results from [7] show it is experimentally possible to compress $z \ll m$ while maintaining a small false positive rate. The savings applied to z are accomplished by reducing the number of k and increasing m, thus requiring more physical memory, but less computation since fewer hashes are used.

In tables 3.1 and 3.2, the transmission bits per element z/n cannot be made arbitrarily small for a given false positive rate. This is in contrast to the independent-subset method

Array bits per element	m/n	16	28	48
Transmission bits per element	z/n	16	15.846	15.829
Hash functions	k	11	4	3
False positive rate	f	0.000459	0.000314	0.000222

Table 3.1. Maintaining a constant compression size z.

Array bits per element	m/n	16	37.5	93
Transmission bits per element	z/n	16	14.666	13.815
Hash functions	k	11	3	2
False positive rate	f	0.000459	0.000454	0.000453

Table 3.2. Maintaining a constant false-positive rate.

in which we could, if necessary, append just one authentication bit to each message and still achieve any desired level of security. But there is some compensation for this loss of efficiency; using Bloom filters, the loss of some messages will not prevent the authentication of other messages.

Chapter 4

Hybrid Authentication

4.1 Overview

Techniques for low-power, low-bandwidth authentication must be able to handle a relatively high number of dropped messages. This presents a problem, since authentication data for one message must be broken up and distributed over many other messages, and some of this authentication data will be lost. The independent-subset and Bloom-filter methods address this problem by introducing redundancy in the authentication data. Here we propose a different approach: a hybrid system which utilizes one-way communication for sending a set of messages, then briefly switches over to slower, more expensive two-way communication in which the receiver can notify the sender which messages were lost in transit, in turn receiving authentication for those messages which were successfully broadcast. The two-way part of the communication is protected by acknowledgements, timeouts, and retransmits to guarantee message delivery. Such a system allows for some messages to be lost while still authenticating the remaining messages within a reasonable amount of bandwidth and time, because two-way communication is mostly avoided. Note that in the most tightly constrained systems, the required two-way communication might be too slow or even impossible; but when it is applicable, this hybrid system could reduce the number of authentication bits per message, since we can use a single fixed-length authentication code to authenticate many messages.

We define two modes for hybrid authentication: sending a stand-alone authentication block or, instead, appending small segments of authentication to future data.

4.2 Stand-Alone Block Authentication

4.2.1 Application Data Transmission

For the first leg of communication, $Node_A$ streams a number of messages, each appended with a small numeric identifier, to $Node_B$. For each message sent, $Node_A$ must store the message and associated identifier for later use during authentication. However, not all the messages sent by $Node_A$ are guaranteed to reach $Node_B$. Over time, there will be a build up of messages stored in memory for both $Node_A$ and $Node_B$. During this period, m

messages will have been sent by $Node_A$ while r messages will have been received by $Node_B$, where $m \geq r$. One-way communication continues until a request for authentication is transmitted.

4.2.2 Authentication Data Transmission

Once $Node_B$ receives a threshold level of n messages, it will initiate two-way communication requesting $Node_A$ to send authentication data. $Node_B$ will send the list of all message identifiers received ID_R allowing $Node_A$ to determine which messages were lost in transit. Until $Node_A$ receives this request, it will continue to stream application data. As this is a two-way communication stream, should $Node_B$ not receive a reply within time t, it will re-send the authentication request. Time delay or message failure will result in new messages being sent to $Node_B$ that were not included in the list of for which authentication was originally requested. $Node_B$ can either store these messages for a future authentication session or, if the previous request was not answered in time t, send an updated list.

Upon successfully receiving the authentication request, $Node_A$ will reply with the authenticator $HMAC_K(messages\_received)$ of every message that was successfully received by $Node_B$. Should $Node_A$ not receive an acknowledgement within time t, it will re-send the authentication data. Once this authentication data has been received, one-way communication will continue as before with $Node_A$ streaming new application data. Should $Node_B$ receive more authentication data rather than new application data after time t, the termination request was not received and must be resent. Once authentication completes, $Node_A$ is free to remove messages and identifiers from its memory.

4.2.3 Stand-Alone Communication Outline

Begin One-Way Communication

- $Node_A$ sends $Node_B$ messages of the form $(message||identifier)$, until $Node_B$ has received a threshold of n messages.

Begin Two-Way Communication

- $Node_B$ initiates authentication by sending (ID_R), the set of identifiers successfully received.
- After time t, $Node_B$ will re-send this request if it has not yet received a reply.
- $Node_A$ replies with $(HMAC_K(M_R))$, the authentication tag for the messages received by $Node_B$.

- After time t, $Node_A$ will re-send this reply if it has not yet received an acknowledgment.

- $Node_B$ replies with an acknowledgement once all authentication data is received, two-way communication ceases and one-way communication resumes.

- After time t, if $Node_B$ received more authentication data rather than new application data, it will re-send the termination request.

4.2.4 Performance

The total time it needed to authenticate a group of n messages is equal to the total time to transmit the n messages along with the time to transmit the authentication request and $HMAC$ block, with possible faults. The total memory required for the process includes m messages stored by $Node_A$ and n messages stored by $Node_B$ along with space on both nodes for the $HMAC$. Recall that $Node_A$ has to store every message and identifier sent, m, while $Node_B$ can only store those received, r. For a fault-prone channel, if $Node_B$ does not request authentication before $Node_A$'s memory fills, there is the chance of overflow. Thus, so far we have assumed the fault rate is low enough so the total number of messages stored by the sender m does not exceed the node's memory.

Though the current scheme does not prevent overflow, the value of n can be modified for any given environment to reduce the risk of data overflow; a smaller n would be more favorable to a nosier channel as the number of messages likely stored by $Node_A$ would be reduced, while a larger n could be supported in a clear channel as the discrepancy between m and r would be reduced. While this method provides a probabilistic defense against overflow, to fully prevent the problem $Node_A$ should cease one-way communication once it has reached or is within its memory limit. At this point, $Node_A$ would initiate two-way communication by requesting the set of identifiers received rather than waiting for $Node_B$ to present the identifiers upon reaching n messages. This places more overhead on two-way communication and may lead to authenticating a set of messages smaller than n, but guarantees that any messages sent and received will be used rather than be invalidated through $Node_A$ overwriting them during overflow, preventing $Node_A$ from using that message in the $HMAC$.

Compared to pure one-way communication, there is a limited amount of overhead involved in sending authentication data via two-way communication, depending on the size of n. However, the hybrid approach overcomes the problem of one-way communication not scaling well to numerous faults. Increasing the number of messages authenticated within a block will reduce the total number of two-way transmissions as the size of the $HMAC$ is independent of the number of messages being authenticated . Overall, stand-alone authentication operates as a light weight protocol for authenticating messages in a low-bandwidth environment with a reasonable authentication delay that depends only on the length of the $HMAC$ and the noise of the channel which may result in re-sends. For information networks where a constant update of data is necessary and down-time to send a stand-alone authentication block is unacceptable, a segmented authentication scheme would be preferred.

4.3 Segmented Authentication

In the previous scheme, $HMAC(M_R)$ was sent all at once, using a two-way communication protocol to guarantee delivery. No data is being sent while the authenticator is being sent. The alternative is for $Node_A$ to split $HMAC(M_R)$ into short segments of length l and to append these segments to successive messages. The special two-way communication ends when $Node_A$ has acknowledged receipt of ID_R. So now there is no guarantee that $Node_B$ will receive all bits of $HMAC(M_R)$, since some messages will be dropped. Thus the threshold n must be set large enough so that, when $Node_B$ has received n messages and nl authentication bits, these nl bits provide the desired level of security.

Each of the $HMAC$ segments sent by $Node_A$ must be unique and independent, so that any set of n segments will provide nl bits of security against forged messages. Simply splitting a long $HMAC$ into shorter pieces does indeed give us a collection of independent functions (since by assumption $HMAC$ is pseudorandom), but this is not sufficient; we do not know how many messages $Node_A$ will have to send before n messages have been received, so $Node_A$ might run out of authentication bits. Achieving an arbitrary number of independent $HMAC$s can be accomplished by including the message identifier with the $HMAC$ such that the authentication segment sent with m_i is the first l bits of $HMAC_K(M_R||i)$.

4.3.1 Segmented Communication Outline

Here is a more explicit list of the steps in the segmented authentication protocol. **Begin One-Way Communication**

- $Node_A$ sends $Node_B$ messages of the form $(message||identifier)$. As there is no guarantee $Node_B$ receives a message, discretion for when to begin authentication is left to $Node_B$, starting after $Node_B$ reaches a threshold of n messages.

Begin Two-Way Communication

- $Node_B$ initiates communication by responding with $(OpAuthReq||ID_R)$, the authentication request operation code followed by the set of message identifiers received.
- After time t, $Node_B$ will re-send the request if it has not yet received a reply.
- $Node_A$ acknowledges receipt of the previous message set M_R.

- After time t, $Node_A$ will re-send the acknowledgement if it receives another $OpAuthReq$ packet, meaning the previous message reply was lost.

Begin One-Way Communication

- After $Node_A$ successfully acknowledges the authentication request, it sends $(message||identifie$ the unique hashed authentication of the previous messages received appended to a new message which is not authenticated.

Begin Two-Way Communication

- Once $Node_B$ has received q $HMAC$s to authenticate the entire set of messages M_R it sends $(OpAuthReq||ID_{new\_R})$, a new authentication request for the set of messages received between the last M_R and the current message.
- After time t, $Node_B$ will re-send the request if it has not yet received a reply.
- Once $Node_A$ successfully receives and acknowledges the request, the protocol will loop between one-way and two-way communication.

4.4 Throttling Authentication

Though the block size of stand-alone authentication and the nl-bit security outlined of segmented authentication were both assumed to be constant, there is no reason that $Node_B$ could not vary the level of security required. For the stand-alone scheme, $Node_B$ can send an *authenticationLength* variable to $Node_A$ when authentication is requested. Similarly, in segmented authentication $Node_B$ can vary the size of n. $Node_A$ will continue sending authentication for M_R until it is notified by $Node_B$ to stop. The ability to throttle the level of authentication could prove valuable to applications which can classify data as critical or insensitive, analyze power remaining, or compare data to other reports and require varying authentication based upon those variables. In such a system, unimportant data can require weaker authentication data while critical information can require greater than usual authentication.

Chapter 5

Signal-Strength Authentication

5.1 Introduction

Usually one wishes to filter out extraneous weak signals, considered noise, from a strong signal. In contrast, the circuit described here allows for the filtering of strong signals that are assumed to have come from adversaries. Sensors in a low-power WSN will have fairly weak signal strength when communicating amongst themselves. Any adversary masquerading as a sensor or otherwise trying to disrupt the network will probably be at a fixed location, and will need to broadcast strong signals in order to reach a large number of sensor nodes, or to reach a node whose location is unknown. An adversary who is restricted to using standard hardware, such as an ordinary laptop, will likewise be broadcasting a stronger signal than what is broadcast by a special-purpose low-power sensor node.

The circuit itself filters out a strong signal, assumed to be malicious, while passing the weak signal along. The hardware is very simple and uses very few circuit elements: AND gates, NOT gates, two resistors, and possibly an amplifier.

There are several assumptions made that allow for this circuit to be useful:

- First, the design assumes that the attackers are unaware of the circuit's existence, or that they do not understand how to circumvent it. Because of the weak signal strength of sensor nodes, it is unlikely that the adversary will become aware that sensors are not receiving its messages, as it cannot receive a reply to confirm the receipt. Of course, if the adversary is nearby sensors and could receive messages, then perhaps she will realize the sensors ignore her messages.
- Second, the Base Station for the sensor nodes uses a different frequency from the sensors themselves. If this is not possible then the architecture must be either expanded to make primitive checks for whether the strong signal is from a friendly base station or the design must be discarded. Some methods for separating messages from a base station and from a sensor may include ensuring a parity for all messages from a sensor. For example, all sensors will always use an even bit parity versus the base station which will always use odd bit parity.
- Third, these laptop-class attackers are attempting to "play by the rules"; the at-

tackers are attempting various attacks that are not blatantly malicious, such as wormhole attacks as opposed to denial of service attacks. A possibly undesirable result of this assumption is that a signal received from a sensor at the same time as a laptop-class attacker's signal will both be discarded.

- Fourth, the incoming digital signals have no DC component. This is a small assumption and easily fixed in circuitry, but worth noting.

5.2 The Circuit

The circuit is comprised of 2 logic gates and an attenuator. The circuit expects digital signals with power levels intact. A diagram of the basic circuit is displayed below in Figure 5.1:

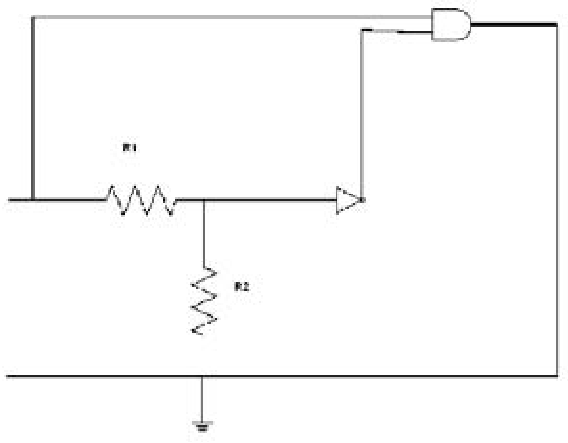

Figure 5.1. Circuit Diagram.

In the case of a high amplitude signal, the attenuator will lower the amplitude of the signal by a certain carefully chosen proportion, but not below the input threshold of the NOT gate. The intention is to maintain the logic value of the waveform across the attenuator for all points. When the signal passes through the NOT gate, the signal is made into its complement. The end result is the complement of the signal and the signal itself are put through the AND gate. Naturally, this results in all 0's, or false, thus removing the signal.

In the case of a low amplitude signal, the signal will pass through the attenuator which is guaranteed to reduce the signal strength below the threshold of the NOT gate. The signal, after leaving the NOT gate, will be all 1's as a result. The consequence of the signal itself going through the AND gate with all 1's is the original signal itself.

5.3 Broadcasting High Power Messages

The strength of this hardware is that it automatically discards messages that claim to be from sensor, but have power levels too high to have been sent by a legitimate sensor. But because of the nature of wireless communication, this architecture does not entirely secure a sensor from laptop-class attackers. The power level of a signal deteriorates at approximately [8] $\frac{1}{R^2}$. This results in some subsection of the circular propagation of the malicious signal having the correct power level of a neighboring sensor. This will result in attackers being able to target *some* nodes, but possibly not knowing whom they are targeting.

5.4 Countering Specific Attacks

The design helps prevent a wide array of attacks on the sensor network. Many of the most devastating attacks can be carried out only by laptops or similarly powerful adversaries [3].

5.4.1 Worm Hole

The Worm Hole attack is very dangerous to sensor networks due to the nature of their communication; all traffic with data must reach the base station. A laptop can take advantage of this by making itself appear to be best place to forward messages from a victim node. Once communication is established, the attacker can either stop communicating all messages completely or pick choice sensors to relay during a time of crisis in which the sensors' data are most needed.

It is worth noting that certain authentication schemes can render this attack useless as the inability of the laptop-class attacker to properly respond to messages from the distant sensors will render it considered as a "failed" sensor.

The hardware design defends against this attack by disallowing the attacker to communicate with nearby sensors because of the strength of its signal. Distant node may also view the attacker as a laptop as well, depending on their distances. Even if the attacker is successful in creating a wormhole, she will be unable to maintain it assuming sensors that invalidate the attacker's messages will notify the base station of the attacker's presence. The affected sensors will simply discard the communication path upon learning the attacker's keys are invalidated.

5.4.2 HELLO floods

A HELLO flood attack sends to nodes a constant stream of HELLO messages, which request key negotiation or simply “notify” the sensor of its presence.

The architecture does extremely well against countering this attack, provided the attacker is not aware of it. The circuit will filter out all of these HELLO messages from a laptop-class attacker immediately, thus saving time spent authenticating and legitimizing these requests.

5.4.3 Sinkholes

Sinkhole attacks are particularly devastating to networks, as a compromised node advertises itself as a single high quality route to the base station [3]. After most nodes route their information through this compromised node, simply turning it off or selectively forwarding packets during critical times can render the sensor network useless.

Sinkhole attacks are perpetrated by high power sensors capable of reaching the base station in generally one hop. These must advertise themselves as a sensor close the base station. This attack can cause interesting effects on sensor networks including most sensors in the network attempting to contact the sinkhole sensor but failing to reach it. Naturally, most of these sensors will see the messages broadcast by the sinkhole as malicious. These will discard it and forward to the base station the keys invalidated; sensors later will ignore this sinkhole.

5.5 Unresolved Concerns

The circuit does not take into account many high-level concepts in signal processing. These concepts include Impedance Matching and Power Level Preservation across demodulators. Additionally, signals from legitimate sensors that are received at the same time as malicious laptop-class attackers will be discarded.

The most fundamental way to bypass this defense, assuming the adversary is aware of it, is to change the power levels of the output signal so the nodes reached by the adversary receive sensor-like power levels for the message. With some basic changes to the hardware, a sensor will process messages that the circuit would have ignored, but will automatically assume them to be malicious. The sensor can then propagate a message to the base station to broadcast a notice that the malicious node is acting improperly and should be ignored. In this way, even if the circuit is bypassed, it becomes easy to track malicious nodes and keys that have become compromised.

References

[1] Mihir Bellare, Ran Canetti, and Hugo Krawczyk. Message authentication using hash functions: the HMAC construction. *CryptoBytes*, 2(1):12–15, Spring 1996.

[2] Burton H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Commun. ACM*, 13(7):422–426, 1970.

[3] Chris Karlof and David Wagner. Secure Routing in Wireless Sensor Networks: Attacks and Countermeasures. *Elsevier's AdHoc Networks Journal, Special Issue on Sensor Network Applications and Protocols*, 1(2–3):293–315, September 2003.

[4] Raymond A. Kristiansen. On the aperiodic autocorrelation of binary sequences. Master's thesis, University of Bergen, 2003.

[5] Rudolf Lidl and Harald Niederreiter. *Introduction to finite fields and their applications.* Cambridge University Press, New York, NY, USA, 1986.

[6] Jr. Marshall Hall. *Combinatorial theory (2nd ed.).* John Wiley & Sons, Inc., New York, NY, USA, 1998.

[7] Michael Mitzenmacher. Compressed bloom filters. In *PODC '01: Proceedings of the twentieth annual ACM symposium on Principles of distributed computing*, pages 144–150, New York, NY, USA, 2001. ACM Press.

[8] Peter Sholander. RF propagation models. Document on RF Propogation and Deterioration with distance, April 2007.

DISTRIBUTION:

2	MS	9018	Central Technical Files, 8944
2	MS	0899	Technical Library, 9536
1	MS	0123	D. Chavez, LDRD Office, 1011