Adaptive Traitor Tracing with Bayesian Networks

Philip Zigoris
Department of Computer Science
University of California, Santa Cruz
Santa Cruz, California 95060
zigoris@soe.ucsc.edu

Hongxia Jin
Content Protection Group
IBM Almaden Research Center
San Jose, California 95120
jin@us.ibm.com

Abstract

The practical success of broadcast encryption hinges on the ability to (1) revoke the access of compromised keys and (2) determine which keys have been compromised. In this work we focus on the latter, the so-called *traitor tracing* problem. We present an adaptive tracing algorithm that selects forensic tests according to the *information gain* criteria. The results of the tests refine an explicit, Bayesian model of our beliefs that certain keys are compromised. In choosing tests based on this criteria, we significantly reduce the number of tests, as compared to the state-of-the-art techniques, required to identify compromised keys. As part of the work we developed an efficient, distributable inference algorithm that is suitable for our application and also give an efficient heuristic for choosing the optimal test.

Introduction

In 1996, the first DVDs were released just in time for the Christmas season. A relatively weak copy protection system, the Content Scrambling System (CSS), was hastily put in place in order to stave off digital piracy. Under CSS, every device manufacturer is assigned a single, confidential key that is embedded in the devices they produce, allowing those devices to descramble the contents of a DVD. (Lotspiech 2005)

In 1999, a 16 year old Norwegian reverse engineered a software player and exposed the first of 400 manufacturers keys. The remaining 399 keys were exposed and published on the internet within weeks. In every sense, the cat was out of the bag. (Lotspiech 2005)

As bandwidth has increased and file sharing on the internet has become rampant, the stakes are even higher than before. Swearing not to make the same mistake twice, the entertainment industry has invested quite a bit in new *broadcast encryption* frameworks. There are two key features in broadcast encryption: first, it is possible to revoke access at the level of an individual player and, second, there is a method for identifying compromised keys in use in illegitimate players/decoders. Coupled, these features make it impossible for someone to produce software or hardware that is not sanctioned by the licensing agency.

Copyright © 2007, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

Of interest here is the second task, identifying compromised keys, which is referred to as *traitor tracing*. Initially, there was interest in tracing algorithms that treat the illegitimate player, or *clone box*, as a stateless black-box, avoiding the need for any kind of reverse engineering. Traitor tracing is done by constructing informative tests that are submitted to the box, of which the output provides some information about which keys the clone box contains. Existing tracing algorithms, while efficient in principle, may require many years of operation for realistic scenarios. In practical terms, the clone box will have effectively defeated the tracing strategy. We will describe this process in more detail in the next section.

As we will also see, we can, in some sense, deduce the best adversarial strategy for the clone box. That is, given a series of tests, what is the response of the clone box that will maximize the number of tests required for the tracing algorithm to succeed. Assuming we know the clone box's strategy, we can leverage this extra bit of information to greatly accelerate tracing. In this sense, ours is not truly a black-box tracing algorithm but it represents a significant improvement over the existing technology.

Underlying our approach is a Bayesian Network used to explicitly represent our belief that certain keys are compromised. Not only does this allow for accurate diagnosis, we can also quantify how informative a test is and use this to guide the tracing process. In this paper we present a scalable inference algorithm that takes advantage of sufficient statistics in the model as well as an effective heuristic method for selecting the optimal test.

Finally, we demonstrate, empirically, that our method offers a dramatic reduction in the number of tests needed for tracing, as compared to the state of the art tracing algorithms.

NNL Traitor Tracing

In this section we present a simplified version of the traitor tracing procedure given by Naor et. al (Naor, Naor, & Lotspiech 2001).

We formalize traitor tracing as follows: let be the set of *valid device keys* and the clone box owns a subset $\subseteq$ of these keys. Let and . The set is, of course, unknown to the tracing algorithm. The task is to determine, within some specified confidence ϵ, at least

Algorithm 1

```
1: NNLTrace( a b )
2: if − then
3:   return
4: else
5:    ← ⌈ + / 2 ⌉
6:    ←  +1
7:   if c − a ≥ c − b then
8:     return NNLTrace a c
9:   else
10:    return NNLTrace c b
```

one key ∈ that is owned by the clone box. That is, ∈ − . We can then invalidate key , i.e. remove it from , and reiterate the tracing procedure.

Broadcast encryption works by first encoding the media, e.g. HD movie, with some key , referred to as the *media key*. Then, the media key is encrypted separately with each of the valid device keys in . As long as a device owns a valid key, it can decrypt the media key and then decrypt the media.

To construct a test, we *disable* a subset of the valid device keys. We do this by encrypting a random bit string instead of . The remaining device keys are said to be *enabled*. Now, if all of the device keys owned by a device are disabled, there is no way for it to recover the media. However, built in to the protocol is a method for the device to validate the media key, meaning that it can recognize when it is under test if it contains a disabled key. Therefore, if a device owns disabled and enabled keys, it can respond strategically; it has a choice whether or not to play/decrypt the media and it can respond non-deterministically. We will say more about this shortly.

From here on, a test can be thought of as simply a set ⊆ of keys. The keys in are enabled and the keys in are disabled. Let be the probability that the clone box plays test . If the probability of playing two tests, and ′, are not equal then it must be case that the clone box owns a device key in the exclusive-or of the two sets. This motivates *NNLTrace* (Algorithm 1), a binary-search-like method for identifying a compromised key. We initially call the procedure with the arguments . The algorithm proceeds by progressively reducing the interval in which we know there must be a compromised device-key. It does this by determining the probability that the midpoint test plays and recursing on the the side with the larger difference in the endpoint probabilities.

The challenge in this style of tracing algorithm is that we must estimate c by repeatedly submitting to the clone box. The smaller a − b , the more tests that are needed to confidently decide which is the larger subinterval. This is advantageous to the clone box. However, the subset tracing procedure always recurses on the larger subinterval and so at every step the gap is reduced by, at most, a factor of 2. This gives us the following lower bound on the gap on the last iteration:

$$- \quad +1 \geq \frac{}{\log_2} \quad —$$

This lower bound is achieved by the *uniform choice* strategy: the clone box tries to decode the media key with one of its device keys chosen uniformly at random. If the key chosen is enabled then the clone box successfully plays the test. If it disabled then it does not. Formally,

$$\text{plays} \quad \frac{\cap}{——}$$

Previous empirical work also substantiates the claim that this is the best clone box strategy (Zigoris 2006) and it will remain our focus in subsequent sections.

Naor et. al show that, regardless of the strategy, the number of tests required for NNLTrace to succeed with probability at least $-\epsilon$ is upperbounded by $^2 \frac{\log}{\epsilon} \ ^3$. However, a clone box can take measures to delay its response to each test and for realistic scenarios, which include details beyond the description given here, we can expect this tracing algorithm to take upwards of 15 years to "defeat" a clone box. One would hardly call this a success.

Traitor Tracing with a Known Strategy

One shortcoming of the algorithm from the previous section is that it cannot take advantage of any prior knowledge about the clone box's strategy. It is by leveraging this information, using the machinery of Bayesian networks and the insights of information theory, that we will be able to demonstrate significant gains over NNLTrace.

First, we introduce some additional notation. Denote by the random variable associated with the outcome of a test or the set of enabled keys comprising a test. The intent should be clear from the context. Let be the set of possible responses to a test. In previous sections the set of responses was limited to the set ; it either plays or it does not. Our approach can accommodate a richer set of responses, a possibility we will study more closely in the experimental section. Associated with each key ∈ is a binary random variable ; is meant to imply that ∈ . We refer to these random variables, collectively, as . In many contexts it is useful to treat as the set of keys for which . That is, represents our belief about which keys are in the clone box.

The question we would like to answer is: provided with a set of test-response pairs ¯ ¯, where are independent (the clone box is stateless) but not identical, what is the posterior probability that the clone box contains a particular set of keys . Applying Bayes rule we have:

$$\bar{} \quad \bar{} \qquad \frac{\bar{} \quad \bar{}}{\bar{} \quad \bar{}}$$

$$\prod \quad \frac{}{\bar{} \quad \bar{}}$$

where the second equality derives from the fact that the results of tests are independent conditioned on . The term is specified by the uniform choice strategy.

The term specifies our belief, *prior* to any observations, that the clone box contains a set of keys . Without any background knowledge we choose the uniform distribution. It is possible to embed domain specific knowledge in

the prior to give the process a "head start". For example, if we knew that a particular subset of devices were more likely to have their keys compromised then we could increase the prior probability that the clone box contained one of those keys.

The denominator $\bar{\ } \bar{\ }$ is the marginal probability of observing the responses to the set of tests and is defined as:

$$\bar{\ } \bar{\ } \sum_{\subseteq} \bar{\ } \bar{\ }$$

The sum is taken over a set of size , making this a difficult quantity to calculate. We will address this issue later. For now, we will assume that it can be calculated efficiently and exactly.

The basic procedure is specified in Algorithm 2. We iteratively choose tests to submit to the clone box and update our beliefs about the contents of the clone box. If at any point we can diagnose a compromised key (line 6) we return the key as well as our current set of beliefs.

Algorithm 2 Strategy based traitor tracing procedure

IGTrace( )
if response of clone-box to is 0 **then**
  return $\emptyset$ *//in this case,* $\emptyset$
loop
  for all $\in$ **do**
    if $- \epsilon$ **then**
      return
  select a test
  submit test to clone-box and get response
  $\leftarrow$ $\bar{\ } \bar{\ }$

Test Selection with Information Gain

We can imagine the testing process as a *decision tree* where every node specifies the next test to submit to the clone box. The response of the clone box dictates the subtree on which to recurse. Every leaf in the decision tree will have a key associated with it and reaching it in the recursion implies the associated key has been compromised. Ideally, we could minimized the expected number of tests needed by mapping out an entire testing strategy. However, this task was shown to be NP-Complete (Hyafil & Rivest 1976).

Instead of devising such a decision tree at once, we take a greedy approach to test selection. First, we quantify the uncertainty about as the *entropy*:

$$\bar{\ } \bar{\ } - \sum_{\subseteq} \bar{\ } \bar{\ } \ {}_2 \ \bar{\ } \bar{\ }$$

We measure the quality of a new test as the *mutual information* [1] between the and . This is equal to the expected reduction in entropy by seeing the result of test . This expectation is taken with respect to the marginal probabilities of each of the possible responses $\in$. Note that it is possible that the marginal probability $\bar{\ } \bar{\ }$ is very different than the true probability of response , . For more background on information theory see, e.g., (MacKay 2003).

Now we can view test selection as the following optimization problem:

$$^{*} \ \operatorname{argmax}_{\subseteq} \ \bar{\ } \bar{\ }$$

The maximum is taken over possible tests and we know of no algorithm for efficiently solving this problem. It is necessary to approximate this by only considering a small subset of the possible tests. To do this, we maintain a set of tests called the *retention set*. At every iteration we try adding a new key to each test in , creating a new set of tests $'$. We then update to be the top tests in $'$. If after an iteration the retention set does not change, then we return the top test in . The total number of tests evaluated by this procedure is $^2 \ ^2$ (Zigoris 2006). Technically, evaluating the gain of each test is exponential in the number of device keys , but we will introduce a method for approximating this quantity.

Related Work

Information gain is a popular criteria for selecting tests in adaptive diagnosis tasks. Rish, et. al (Rish *et al.* 2005) apply a method similar to ours to the task of diagnosing faults in distributed system. There are a few key differences. First, they choose tests from a small pool whereas we are faced with an exponential number of tests. Secondly, they assume there is only a single fault in the system whereas in our work there can be multiple faults, i.e. compromised device keys. They also use an approximate inference algorithm, mini-buckets, whereas we will devise an efficient and exact inference algorithm for our model.

Bayesian networks and information gain have also been applied to a probabilistic version of the game of Mastermind with reasonable results (J. 2004). Mastermind is a popular board game created in the 70's where two players engage in a query-response loop. In the original formulation, game play is completely deterministic. The goal is for one player, the code-breaker, to identify the code put down by the code-maker. In many ways, it is similar to the task we are face with. We can imagine the clone box as a secret code that we are trying to reveal. In our game, though, we are only required to find one bit in the 'code'.

Representation and Inference with Bayesian Networks

So far we have avoided any discussion of how to compute the necessary probabilities for our algorithm. There are two points to keep in mind. The first is that an erroneous diagnosis, which results in the disabling of a device, is very costly. This imposes the constraint that the marginal probability that must be calculated exactly. Therefore, we cannot rely on approximate inference algorithms such as loopy belief propagation (Pearl 1988) or mini-bucket (Dechter 1997). However, test selection, which depends on the full

[1] Sometimes referred to in the machine learning literature as the *information gain*

joint distribution of , is already approximate so its reasonable to approximate the joint distribution. In this section we present an efficient (polynomial time) inference algorithm that does exactly these two things.

It helps to visualize our statistical model as a Bayesian Network. For our application, the nodes in our network correspond to the tests and the keys in the frontier. As a first approach, consider complete bipartite network in Figure 1. The conditional probability distribution (CPD) for each test is specified by the clone box strategy. For every key we must specify the prior probability that . Without any background information, we set this to 0.5. However, in practice we may have reason to believe that some keys are more likely to be compromised. Or, if we expect that , then we set —.

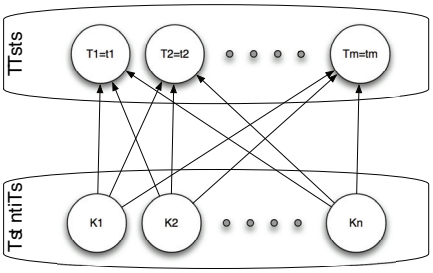

Figure 1: A simple bipartite network for our task. Every variable in the test layer is conditioned on every variable in the frontier.

The problem with using this representation is adding a test result as evidence creates a dependency between the key nodes. Thus, in order to find the marginal probability for one key in the frontier, standard inference algorithms, such as variable elimination (Dechter 1999) and junction tree (Huang & Darwiche 1996), must create intermediate factors whose table sizes are exponential in .

For arbitrary clone-box strategies, we can not expect to do better than this. However, the uniform choice strategy had a very concise expression; it depended only on the ratio of enabled to disabled compromised keys. It does not depend on the specific keys that are compromised but only on how many are enabled and disabled.

We can exploit these sufficient statistics to gain efficiency in our model. The modified network is illustrated in Figure 2. Let ∈ be the set of keys enabled in ; similarly, let ∉ be the set of disabled keys. Form two binary trees with and as the leaves of each and with edges pointing towards the root. Denote, respectively, the roots of these trees as and ; connect an edge from both and to . Each of the internal nodes is simply the sum of its parents: Δ $\sum_{\in(\ )}$. The CPD of is now $\frac{}{+}$. Note that in tabular form the CPDs are ^{2} in size.

We now present a specific method for storing and updating beliefs that is particularly well-suited to our application. In order to approximate the joint distribution of we first partition the frontier into disjoint sets such that ^{i} is a reasonable size for all , allowing us to store as a table in memory. We approximate our belief as

$$\approx \prod$$

which is equivalent to assuming the partitions are independent. If and are independent random variables then

where is the entropy function introduced in the section on test selection. Whereas before, calculating information gain, and implicitly entropy, required us to sum over terms, we can now decompose the calculation into a sum of sums over ^{i} terms. Note, too, that since is in memory, calculating ∈ for some key ∈ can be done in ^{i} time. With a simple caching strategy, this can also be reduced to constant time.

Algorithm 3 gives the procedure for finding the posterior distribution of a partition given test result . The procedure *getCountDistribution* finds the probability distribution over the number of enabled and disabled keys in a set of partitions. Also, , which appears in the information gain, is calculated as a side effect of the procedure. The terms ∩ and ∩ in line 13 correspond to the number of enabled and disabled keys in , respectively. The bottleneck in this procedure is calculating , which would comprise an ^{2} size table. One nice feature of this approach is that the posterior distribution for each partition can be calculated independently making it easy to distribute across multiple servers.

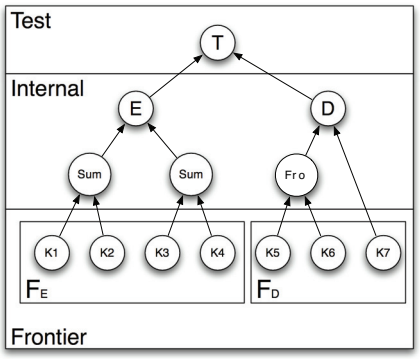

Figure 2: An alternative network structure for the fixed-strategy. The internal nodes act as an interface between the frontier and the test layer, calculating the sufficient statistics and . Reducing the frontier in this way makes efficient inference possible.

Experiments

In this section we present empirical work comparing the number of tests needed by the described methods. All exper-

Algorithm 3 finds and

getPosterior( )
← getCountDistribution /
/
← ∑
← ∑ $_a$
←
return and

getCountDistribution( )
←
←
for all do
← ∑ $_i$ − ∩ −
∩

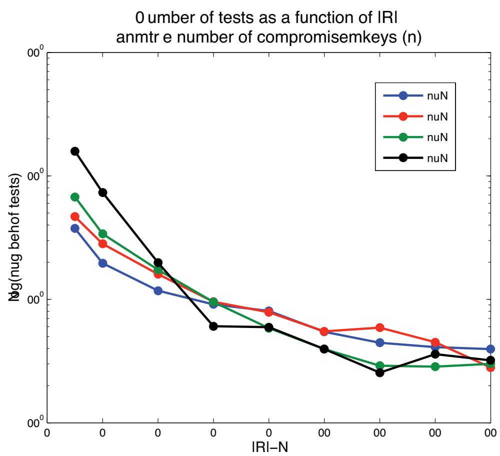

Figure 3: Results of watermarking experiment

iments were repeated 20 times for random choices of compromised keys. In all experiments, we use ϵ and Preliminary experiments showed that the number of tests is not particularly sensitive to , the retention set size, so in all experiments . We only report results for the uniform choice strategy since it is, theoretically, the most challenging strategy to defeat. Preliminary experiments also confirmed this was the case.

All experiments were run using Matlab running under Linux, with 3GB of memory and a 2Ghz Pentium 4 processor. The code was optimized to take advantage of Matlab's efficient matrix operations and with , exact inference and test selection takes around 45 seconds.

Watermarking,

In this experiment we assume there is some watermarking mechanism built into the encryption scheme. That is, every key has a watermark associated with it. If the clone box decrypts the test with then the response to is . This puts the clone box at a disadvantage since it may reveal much more information about its contents. For instance, if each of the keys in the frontier has a different watermark, then we can immediately identify a compromised key when the clone box plays. To clarify, we restate the CPD for test , for / , as:

$$\in$$

For every trial, both the set of compromised keys and the watermark assignment are chosen randomly. The frontier size is fixed at 16, the number of compromised keys is varied from 1 to 8 and − is varied from 1 to 16. The results are plotted in Figure 3. For clarity we omit the standard error bars. Note that the -axis is on a logarithmic scale. The number of tests, therefore, decreases extremely fast. In the limit, the number of tests, for any number of compromised keys, will converge to 1 with since we expect each key to have a unique watermark.

Something to note, but that is difficult to read off of the plot, is that as increases, having more compromised keys becomes beneficial. The reason for this is that the clone box is more likely to contain a key with a unique watermark. Also, we gain more information when a clone box gives a non-zero response. If there only a few compromised keys then many of the responses will be 0 since the test is more likely, at least in the early stages, to disable a large fraction of the clone box keys.

Partition size

This experiment is designed to study the effect of the partitioning size on the number of tests. Again, we vary the number of compromised keys from 1 to 8 and leave the frontier size fixed at 16. The size of the frontier partitions is varied from 2 to 16, corresponding to exact inference. The results are plotted in Figure 4.

Intuition tells us that the number of tests should decrease as the partition size increases. What is surprising is that for the number of tests increases with the partition size. And note that the slope of the other lines seems to increase with . It seems that smaller partitions actually benefit the tracing process when there are a large number of compromised keys. Why this is the case is an intriguing question, one for which we don't have a complete answer. Our hypothesis is that information gain is not always the best optimization criterion for selecting a test. Selection based on information gain represents a greedy step towards minimizing the entropy of . The true aim of our procedure is to diagnosing a compromised key, not minimizing the entropy. Information gain seeks to minimize our uncertainty about all keys, but in actuality we only need to minimize our uncertainty about one key. By finely partitioning the frontier we seem to deemphasize the global nature of information gain and instead focus on diagnosing particular keys.

Overall comparison

Finally, in Table 1 we present a comparison of IGTrace to NNLTrace. It is immediately clear that IGTrace outperforms NNLTrace in all cases. Note, too, that there is substantially less variance in the number of tests needed by IGTrace.

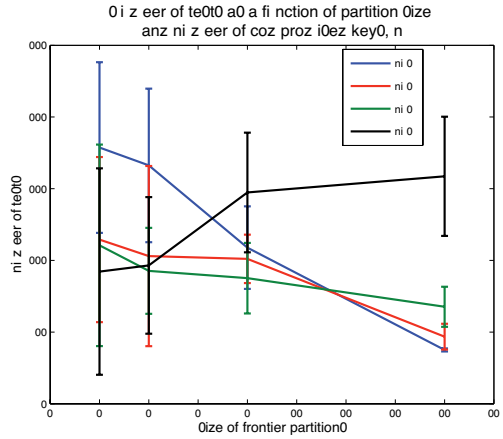

Figure 4: Results of experiment varying partition size.

		2	4	8
	IG	19±8.0	41±12.6	42±16.1
	NNL	605±181.5	919±423.8	1424±160.6
	IG	24±8.4	52±11.1	86±38.7
	NNL	948±299.2	1408±601.2	2632±607.4
	IG	32±9.4	57±18.6	113±36.5
	NNL	1263±415.1	1686±949.3	3562±1223.1
	IG	38±9.5	64±14.7	129±26.1
	NNL	1366±399.4	2326±1221.7	4506±1547.4
	IG	47±9.0	68±14.0	159±41.6
	NNL	1493±478.0	3660±1239.4	4278±1588.7
	IG	57±7.3	86±15.9	150±30.3
	NNL	1845±488.9	4525±1457.6	6229±3199.4

Table 1: Comparative performance of the two tracing methods for a variety of frontier sizes and clone box sizes. Each entry lists the mean and standard deviation of the number of tests. Column headers indicate the number of compromised keys.

Conclusion

This work introduced a new method for traitor tracing when the strategy of the clone box is known and demonstrated, empirically, a significant improvement over existing black-box tracing methods. We took advantage of the specific functional form of the uniform choice strategy to devise an efficient, and distributable, inference algorithm. This algorithm does not sacrifice any accuracy in diagnosis and only approximates the test selection step.

A surprising observation was that the number of tests does not necessarily decrease as the partition sizes increase. This seems to suggest that information gain is not the best criteria for test selection and it may be worth exploring other criteria for test selection. Nevertheless, it seems to work very well and offers a dramatic improvement over the other methods.

Our work does beg the question: how does one obtain information about the clone box strategy? In the future, it is worth investigating methods for learning this strategy along the way, perhaps by representing the clone box strategy as a latent variable. Similarly, it would also be interesting to investigate the sensitivity of IGTrace to errors in the clone box strategy. We would like it to be such that if the actual strategy deviates from the modeled strategy we are still guaranteed to make accurate diagnoses.

References

Dechter, R. 1997. Mini-buckets: A general scheme for generating approximations in automated reasoning. In *IJCAI*, 1297–1303.

Dechter, R. 1999. Bucket elimination: A unifying framework for reasoning. *Artificial Intelligence* 113(1-2):41–85.

Huang, C., and Darwiche, A. 1996. Inference in belief networks: A procedural guide. *Intl. J. Approximate Reasoning* 15(3):225–263.

Hyafil, L., and Rivest, R. L. 1976. Constructing optimal binary decision trees is np-complete. *Inf. Process. Lett.* 5(1):15–17.

J., V. 2004. Bayesian networks in mastermind. *Proceedings of the 7th Czech-Japan Seminar on Data Analysis and Decision Making under Uncertainty*. 185–190.

Lotspiech, J. 2005. *Digital Rights Management for Consumer Devices*. chapter 23.

MacKay, D. J. C. 2003. *Information Theory, Inference, and Learning Algorithms*. Cambridge University Press. available from http://www.inference.phy.cam.ac.uk/mackay/itila/.

Naor, D.; Naor, M.; and Lotspiech, J. B. 2001. Revocation and tracing schemes for stateless receivers. In *CRYPTO '01: Proceedings of the 21st Annual International Cryptology Conference on Advances in Cryptology*, 41–62. London, UK: Springer-Verlag.

Pearl, J. 1988. *Probabilistic reasoning in intelligent systems: networks of plausible inference*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Rish, I.; Brodie, M.; Ma, S.; Odintsova, N.; Beygelzimer, A.; Grabarnik, G.; and Hernandez, K. 2005. Adaptive diagnosis in distributed systems. *IEEE Transactions on Neural Networks* 16:1088–1109.

Zigoris, P. 2006. Improved traitor tracing algorithms. Technical report, UC Santa Cruz.