How To Safeguard Your Sensitive Data

Bob Mungamuru
Department of Computer Science
Stanford University
Stanford, CA, USA
bobji@stanford.edu

Hector Garcia-Molina
Department of Computer Science
Stanford University
Stanford, CA, USA
hector@cs.stanford.edu

Subhasish Mitra
Department of Electrical Engineering
Stanford University
Stanford, CA, USA
subh@stanford.edu

Abstract

In order to safeguard a sensitive database, we must ensure both its privacy and its longevity. However, privacy and longevity tend to be competing objectives. We show how to design a system that provides both good privacy and good longevity simultaneously. Systems are modelled as compositions of two basic operators, Copy *and* Split*. We propose metrics with which to evaluate the privacy, longevity and performance offered by such systems. The search for the "best" system under these metrics is then formulated as a constrained optimization problem. Solving the optimization problem exactly turns out to be intractable, so we propose techniques for efficiently finding an approximate solution.*

1 Introduction

"Keep it secret, keep it safe!"
–Gandalf

Suppose we have a critically important database that must be safeguarded both against unauthorized access and against data loss. This database could contain, say, credit card numbers for our customers, patient records at a hospital, or financial records at a bank. To safeguard such data we probably want to use both replication (for longevity) and encryption-like operations (for preventing theft or unauthorized access).

For example, Figure 1 graphically shows how we might protect a root database r from unauthorized access and data loss. The *Split* operator S represents a "split" of the database into a materialized (i.e., physically stored) *terminal* object a, and a non-materialized (i.e., transient) object f that is to be processed further. Here, both a and f are needed to reconstruct r (e.g., S might encrypt r into ciphertext a using key f). Because we are concerned that the "key" f could be lost, we make two copies, b and e, using the *Copy* operator C. One copy of f is stored as terminal b, perhaps inside the desk of the company's CEO. Since we do not trust anyone else at the company to by themselves hold the other copy e, we split e further into terminals c and d. This way, c and d (e.g., held by the CFO and the CIO) are both needed to reconstruct f.

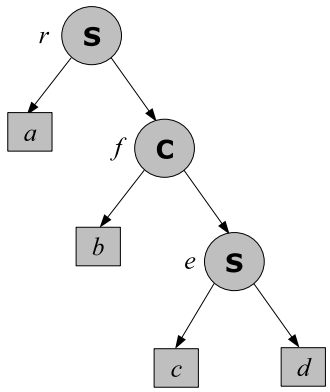

Figure 1. Example of a configuration.

There are of course many other ways one can compose Copy and Split operators – each composition results in a *configuration* with different levels of protection against break-ins and data loss. A configuration may also have other desirable or undesirable properties. In particular, some configurations will turn out to be *unimplementable* because they compose Split and Copy operators in a way that does not "make sense". For example, there may be a circularity in Split operators: the key needed by a Split may somehow depend on the value of the ciphertext it is supposed to generate.

Split and Copy operators (which will be defined precisely in Section 2) may not be the only mechanisms for safeguarding data, but they are by far the most popu-

lar. And even with just these two operators, there is a huge number of possible configurations – many unimplementable, many mediocre, and a small number of very good ones that provide the right (application-dependent) balance between longevity and privacy. In this paper, we address the problem of searching for these very good configurations. To do so, we formally define the concepts that have been introduced here via examples. We define metrics to evaluate configurations, and we present algorithms that efficiently search for the very good configurations.

Our work is distinguished from most of the existing literature, since we simultaneously consider both privacy and longevity. There is a large body of work on systems that ensure longevity of data. This ranges from work on fault tolerance and failure statistics to error correction schemes and RAID [1]. However, in this body of work, privacy issues are rarely considered. On the other hand, there is a sizable literature on data privacy and security, ranging from encryption schemes (e.g., [2]) to secure data storage methods (e.g., [3]). However, in the privacy literature, there is little consideration of data longevity and fault tolerance. There is some work attempting to integrate longevity and privacy [4], which describes how certain encryption schemes might be used to build survivable, secure storage systems. Our work, however, takes a more quantitative approach than [4], by searching for solutions to specific optimization problems over the space of possible systems.

In summary, in this paper we make the following contributions:

- We define metrics for evaluating systems that safeguard data. The metrics quantify the provided levels of privacy and longevity, as well as the semantic correctness and physical realizability of the protection schemes.
- Using these metrics, we formulate the search for a good safeguarding system as a constrained optimization problem.
- We suggest a two-stage strategy for solving this optimization problem. The first stage finds a family of systems that are optimum with respect to a subset of the metrics. The second stage picks out a system from this family, which performs well under the remaining metrics.
- The first-stage problem is itself difficult to solve for large instances. We therefore propose both exact and approximate techniques for its solution.
- We present an experimental evaluation that answers some of the questions about our approach. Are the privacy and longevity gains achieved by our optimization algorithms significant? (Yes.) Are many configurations that provide good protection unimplementable? (Yes.) How well does our approximate scheme do against an exhaustive search? (Very well, in many cases.)

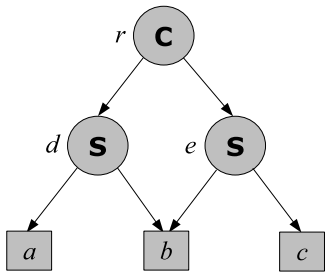

Figure 2. A configuration with sharing.

2 Configurations

We begin with two operators, *Copy* and *Split*, from which safeguarding systems will be built up. A k^{th}-order Copy operator produces as its output k identical copies of its input. A k^{th}-order Split operator breaks its input into k output pieces, such that all k pieces are necessary and sufficient to reconstruct the input.

There are several ways to implement a k^{th}-order Split operator. One way would be to use encryption, where the input is plaintext and the output is ciphertext along with $k-1$ keys. Another option is to randomly generate $k-1$ bit-sequences of length equal to the input, and output these $k-1$ sequences along with the input XOR'ed with all of these sequences. In both of these examples, all k outputs are needed to reconstruct the input. We use Split operators as abstractions of these and other splitting operations.

Systems for safeguarding data can be built by composing Split and Copy operators in interesting ways. Such compositions are referred to as *configurations*. An example was given in Section 1 (see Figure 1). The data we are trying to safeguard is at the root, r. The root is encrypted, say, and the ciphertext is stored at a and the key at f. Two copies, b and e, are made of this key. One of these copies, e, is XOR'ed with a random bit-sequence stored at c, and the result is stored at d. Only the terminal vertices a, b, c and d are actually materialized and physically stored. The non-terminal vertices e, f and r are transient data elements that are by-products of recursively splitting and copying the original data. In particular, observe that the root r is not stored anywhere. Therefore, there is no single terminal vertex that an attacker can "steal" that will allow him to reconstruct the original data.

In general, configurations can be rooted directed acyclic graphs (DAGs). For example, consider Figure 2, where d and e are copies of the root data r. Here, the vertex b is *shared* by both d and e – it represents a single key that is used to encrypt both d and e.

Formally, a configuration Θ is composed of a set of vertices $\mathcal{V}$. This set $\mathcal{V}$ is partitioned into $\mathcal{T}$, the set of terminal vertices, and $\mathcal{N}$, the set of non-terminal vertices. Note that the dependence on Θ of $\mathcal{V}$, $\mathcal{T}$ and $\mathcal{N}$ has been notationally suppressed. In what follows, we assume that the elements of $\mathcal{T}$ are labelled $a, b, c, \ldots$ and so on. We also assume that the root is labelled r. The power set $2^{\mathcal{T}}$ is the set of all subsets of $\mathcal{T}$.

Require: Θ
1: $F_\Theta \leftarrow r$
2: **while** $\exists y \in support(F_\Theta)$ s.t. $y \in \mathcal{N}$ **do**
3: let $\{x_1, x_2 \dots x_k\}$ be the children of y
4: **if** y is a Copy vertex **then**
5: replace y in F_Θ by $(x_1 + x_2 + \cdots + x_n)$
6: **else if** y is a Split vertex **then**
7: replace y in F_Θ by $(x_1 \cdot x_2 \cdot \cdots \cdot x_n)$
8: **end if**
9: **end while**
10: return F_Θ

Figure 3. Pseudocode for constructing the access formula F_Θ.

Corresponding to any configuration Θ is a Boolean expression F_Θ, referred to as its *access formula*. F_Θ may include parentheses (i.e., it is a particular factorization) and is always monotone (i.e., no negation). The *satisfying assignments* of F_Θ, denoted $\mathcal{S}(\Theta) \subseteq 2^{\mathcal{T}}$, tell us which terminals an attacker has to break into in order to reconstruct the sensitive data at the root, r. Conversely, the *falsifying assignments* of F_Θ, denoted $\mathcal{F}(\Theta) \subseteq 2^{\mathcal{T}}$, are those subsets of terminals that, if destroyed, would make our sensitive data unrecoverable.

The pseudocode in Figure 3 describes how to construct the access formula F_Θ from the configuration Θ. For example, for the configuration in Figure 1, $F_\Theta = a(b + cd)$. In this case, an attacker can reconstruct the data at r by breaking into terminal a in addition to either b alone or both c and d. Thus, $\{a, b\}$ and $\{a, c, d\}$ are satisfying assignments of F_Θ. Similarly, $\{a\}$ and $\{b, c\}$ are examples of falsifying assignments. The algorithm in Figure 3 is also "invertible" – given F_Θ, we can reconstruct Θ by starting at the terminals and recursively replacing disjunctions and conjunctions by Copy and Split vertices, respectively. Thus, the correspondence between a configuration and its access formula is one-to-one. As such, we will often represent a configuration Θ directly by its access formula F_Θ.

Additional details on our model are given in [5]. In particular, it is shown that not all possible configurations are semantically correct, or even physically realizable. As an example of incorrect semantics, the configuration $F_\Theta = (a + b)(b + c)d$ uses a third-order Split operator, but as discussed in [5], we can reconstruct its data using just two of its children. The main contribution of [5] was a taxonomy over the space of configurations, comprised of four nested classes: *implementable*, *proper*, *simple* and *read-once*. The most general property (implementability) captures what is required for a configuration to be physically realizable with respect to a broad class of copying and splitting primitives. Thus, in this paper, we will always restrict our attention to those configurations that are implementable, at the very least.

3 Evaluating Configurations

The purpose behind modeling and classifying configurations is to find systems that safeguard our data effectively. However, it is presently unclear how we might evaluate the effectiveness of a given configuration. In particular, how might we quantify the privacy and longevity provided by a configuration? In Section 3.1, we suggest probabilities of failure as a unifying metric to measure both the privacy and longevity offered by a configuration. Then, in Section 3.2, we suggest and motivate other metrics such as depth and class as quantities of possible interest in our search for a good configuration.

3.1 Probabilities of Failure

One approach to quantifying privacy and longevity is to use *failure probabilities*, namely the probability of break-in and the probability of data loss. The *probability of break-in*, $P(\Theta)$, is the probability that an attacker breaks into enough terminals to be able to reconstruct the root, r. Similarly, the *probability of data loss*, $Q(\Theta)$, is the probability that enough terminals are lost that we cannot recover the data at r.

We use the term "break-in" to generally refer to an attacker gaining unauthorized access to our data. For example, if the terminal b in Figure 1 was stored on a disk in the CEO's desk, then a "break-in" might simply mean that the disk is physically stolen. Alternatively, if b is a password-protected network node, then a "break-in" might refer to a cracked or leaked password. Similarly, "data loss" refers to any event causing data to no longer be accessible to us.

We illustrate the concept of failure probabilities using an example. Consider $F_\Theta = ab + bc$, illustrated in Figure 2. Let us assume that each terminal is broken into with probability $\frac{1}{4}$, independent of all other terminals. An attacker wishing to reconstruct r must do one of three things – he must either break into terminals a and b only, or terminals b and c only, or all three of a, b and c. Thus, the probability of data loss will be the sum of probabilities of these three mutually exclusively outcomes i.e., $P(\Theta) = 2\left(\frac{1}{4}\right)^2\frac{3}{4} + \left(\frac{1}{4}\right)^3 = \frac{7}{64}$. Similarly, an attacker must destroy any of the following sets of terminals to cause r to be lost: $\{b\}$, $\{a, b\}$, $\{b, c\}$, $\{a, c\}$ or $\{a, b, c\}$. Assuming the attacker destroys each terminal independently with probability $\frac{1}{4}$, we sum over the probabilities of these five outcomes to find $Q(\Theta) = \frac{1}{4}\left(\frac{3}{4}\right)^2 + 3\left(\frac{1}{4}\right)^2\frac{3}{4} + \left(\frac{1}{4}\right)^3 = \frac{19}{64}$.

The notion of failure probabilities is formalized as follows. Define a pair of independent probability spaces $(\Omega_P, \mathbb{P})$ and $(\Omega_Q, \mathbb{Q})$, which represent an attacker's attempts to reconstruct and destroy our sensitive data, respectively. Ω_P and Ω_Q are referred to as *sample spaces*. The elementary outcomes $\omega \in \Omega_P$ are subsets of terminals that the attacker manages to break into. Elementary outcomes $\omega \in \Omega_Q$ are subsets of terminals that are destroyed by the attacker. Thus, $\Omega_P = \Omega_Q = 2^{\mathcal{T}}$. $\mathbb{P}$ and $\mathbb{Q}$ are discrete probability measures over events in Ω_P and Ω_Q, respectively, so that $\sum_{\omega\in\Omega_P} \mathbb{P}(\omega) = 1$ and

$\sum_{\omega \in \Omega_Q} \mathbb{Q}(\omega) = 1$. From the preceding discussion, we have $P(\Theta) \equiv \mathbb{P}(\{\omega \in \mathcal{S}(\Theta)\})$ and $Q(\Theta) \equiv \mathbb{Q}(\{\omega \in \mathcal{F}(\Theta)\})$.

The physical meaning of $\mathbb{P}$ and $\mathbb{Q}$ is as follows. $\mathbb{P}$ and $\mathbb{Q}$ describe an experiment that lasts a fixed period of time, say, ten years. We wish to answer questions such as: what is the probability that our data will still be available ten years from now? Or, how likely is it that no break-ins occur over the next ten years? The answers to these questions (i.e., $P(\Theta)$ and $Q(\Theta)$) depend on the ten-year security and reliability characteristics (i.e., $\mathbb{P}$ and $\mathbb{Q}$) of the terminals across which our data is distributed. In Sections 4-6, we will solve the following problem: given $\mathcal{T}$, $\mathbb{P}$ and $\mathbb{Q}$, find the "best" Θ. That is, given a set of physical resources, and knowledge of their failure characteristics, what is the configuration that best utilizes these resources?

Finally, although we have assumed in our examples that failures of individual terminals are independent, this assumption is not at all essential. The definitions of $\mathbb{P}$ and $\mathbb{Q}$ are general enough to capture correlations between the failures of terminals. For example, we can capture a situation wherein, say, terminal a being destroyed implies that b will also be destroyed (perhaps a and b are sitting in the same server room). We can also describe negative correlation – say, terminal a being broken-into implies that with high probability b will not be broken-into (maybe the sysadmin receives an alarm, and disconnects b from the network).

3.2 Other Metrics

The failure probabilities $P(\Theta)$ and $Q(\Theta)$ are the most important metrics that will guide our search for good configurations. However, there are others that may be of importance to the designer. We describe some of these metrics here.

3.2.1 Depth

A configuration's *depth*, $D(\Theta)$ is defined as the maximum number of vertices between the root and any of the terminals. The depth is a measure of the processing time needed to compute the terminal data elements from the original data. For example, the configuration in Figure 2 has $D(\Theta) = 2$.

3.2.2 Class

As discussed in Section 2, within the space of all possible configurations, we can identify *classes* that have desirable properties. *Implementable* and *proper* configurations are guaranteed to be physically realizable and irredundant, respectively. *Simple* and *read-once* configurations provide further structural properties. In [5], algorithms are presented for checking whether a configuration is a member of a given class. For example, the configuration in Figure 4 is shown to be unimplementable. Terminals a and b must be equal since they are copies of c, which implies that d is being split into two identical components. However, the latter does not make sense, because it forces to use a "divide-by-2" split, which is not secure. We will always require a configuration to be at least implementable, but sometimes we may wish to impose a stronger restriction. We denote by $\mathcal{C}(\Theta)$ the class of a given configuration.

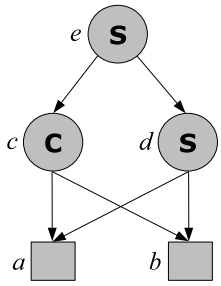

Figure 4. Unimplementable configuration.

3.2.3 Terminals

The *number of terminal vertices*, $M(\Theta)$, is a measure of the physical storage required to deploy the configuration. When we search for good configurations, we will always impose an upper bound on $M(\Theta)$. Recall that in a configuration, only the data at the terminal vertices are materialized. Thus, a bound on $M(\Theta)$ can be thought of as a resource constraint.

3.2.4 Non-Terminals

The *number of non-terminal vertices*, $N(\Theta)$, is a measure of the computational resources required in computing the terminal data elements. It is similar in spirit to measuring depth, although not exactly the same. A Split operator with, say, six children all of whom are Split or Copy operators would have a small depth (i.e., $D(\Theta) = 2$), but would still require seven operators total. Measuring depth alone would not capture this.

3.2.5 Groups

Finally, we may stipulate that certain *groups* must be allowed to reconstruct the data. We refer to these as *allow groups*. For example, we may require terminals a and b to be together sufficient to reconstruct the data. Such a statement is equivalent to requiring that $\{a, b\} \in \mathcal{S}(\Theta)$. We may also stipulate that certain groups, referred to as *deny groups*, be denied the ability to reconstruct the data. For example, breaking into c and d should not be sufficient to reconstruct the root. Such a statement is equivalent to specifying that $\mathcal{T} \setminus \{c, d\} \in \mathcal{F}(\Theta)$ is a falsifying assignment (the '$\setminus$' denotes set difference). As an illustration, one possible configuration that meets these requirements is shown in Figure 1. We assume that the specification of allow and deny groups have no redundancies (e.g., both $\{a\}$ and $\{a, b\}$ are listed as allow groups) and no conflicts (e.g., $\{a\}$ is an allow group but $\{a, b, c\}$ is a deny group).

4 Optimization

We now return to the problem of searching for good configurations. It does not make sense to simply search for the "best" configuration. The best possible configuration for privacy is simply a k-way Split, but it is the worst for longevity. Conversely, the best configuration for longevity is a Copy, but it is worst for privacy. Moreover, we can do arbitrarily well along either of these dimensions by simply using unbounded numbers of terminals! Therefore, a better question to ask would be: subject to some minimum level of privacy, and an upper bound on the number of terminals, which is the configuration that provides us the most longevity? Using the metrics introduced in Section 3, we can write down the following optimization problem:

$$\begin{aligned} \min_{\Theta} \quad & Q(\Theta) \\ \text{s.t.} \quad & P(\Theta) \leq P_0 \\ & \{\omega_0^s, \omega_1^s, \dots\} \subseteq \mathcal{S}(\Theta) \\ & \{\omega_0^f, \omega_1^f, \dots\} \subseteq \mathcal{F}(\Theta) \\ & M(\Theta) \leq M_0 \\ & N(\Theta) \leq N_0 \\ & D(\Theta) \leq D_0 \\ & \mathcal{C}(\Theta) \in \mathcal{C}_0 \end{aligned} \tag{1}$$

Here, P_0 is an upper bound on $P(\Theta)$ that indicates the highest probability of break-in we are willing to tolerate. M_0, N_0 and D_0 are our constraints on the various metrics introduced in Section 3.2. The sets $\{\omega_i^s\}$ and $\{\omega_i^f\}$ are the allow and deny groups, respectively, as described in Section 3.2.5. $\mathcal{C}_0$ is the class that we require our configuration to fall into, as discussed in Section 3.2.2. The set of physical terminals $\mathcal{T}$ and their failure characteristics $\mathbb{P}$ and $\mathbb{Q}$ (which are needed to compute $P(\Theta)$ and $Q(\Theta)$) are known beforehand.

In (1), we are maximizing longevity by minimizing $Q(\Theta)$, the probability of data loss. Note that we could have, instead, maximized privacy (i.e., by minimizing $P(\Theta)$) subject to some minimum longevity requirement. In order to simplify the exposition, we will focus on (1) in this paper.

In principle, this completes the task of finding an optimal configuration. If we could solve (1) exactly, then we would be done. Of course, the solving (1) exactly is extremely difficult. To get a rough idea of how difficult, recall from Section 2 that configurations are in one-to-one correspondence with factored monotone Boolean expressions. Thus, we must exhaustively search through the space of factored monotone Boolean expressions to find a global optimum. Now, even ignoring factorizations, the space of monotone DNF Boolean expressions alone is enormous [6]. For example, for $M_0 = 8$, there are on the order of 10^{22} expressions to consider! Therefore, we must resort to approximate methods to find good solutions to the optimization problem in (1).

A key observation will allow us to devise a strategy for finding approximate solutions to (1). But first, we need a pair of definitions. Two configurations Θ_1 and Θ_2 are said to be *logically equivalent* if F_{Θ_1} can be transformed to F_{Θ_2} by applying the laws of Boolean algebra. We write $\Theta_1 \approx \Theta_2$. Then, we say that a given metric is *logically invariant* if any set of logically equivalent configurations must have the same value for that metric. That is, the value of a logically invariant metric is preserved across logical transformations of F_Θ. For example, the depth $D(\Theta)$ of a configuration is not logically invariant. To see this, consider the logically equivalent configurations $F_{\Theta_1} = a + b + cd$ and $F_{\Theta_2} = a + (c + b)(b + d)$, depicted in Figure 5. $\Theta_1 \approx \Theta_2$, but $D(\Theta_1) = 2$ and $D(\Theta_2) = 3$.

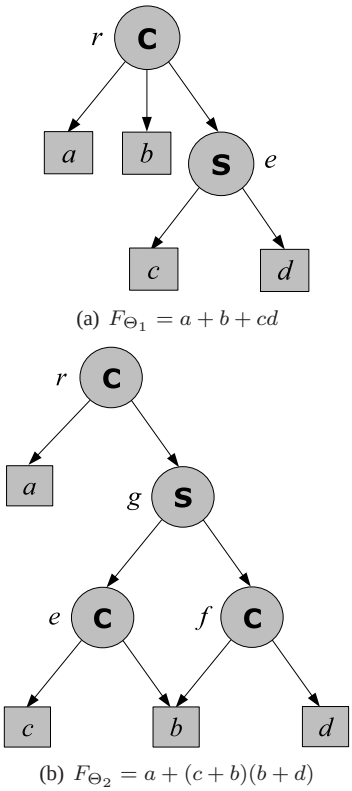

Figure 5. Two logically equivalent configurations, $\Theta_1 \approx \Theta_2$, with $D(\Theta_1) = 2$ and $D(\Theta_2) = 3$.

The key observation is this: $P(\Theta)$ and $Q(\Theta)$ are logically invariant metrics. It is easy to see why. Suppose $\Theta_1 \approx \Theta_2$. $P(\Theta_1)$ is the probability that enough terminals in Θ_1 are stolen for an attacker to reconstruct the root, r. By definition, $P(\Theta) = \mathbb{P}(\{\omega \in \mathcal{S}(\Theta)\})$. However, F_{Θ_1} and F_{Θ_2} are equivalent according to the rules of Boolean logic, which, by definition of satisfying assignments implies $\mathcal{S}(\Theta_1) = \mathcal{S}(\Theta_2)$. Therefore, we conclude that $P(\Theta_1) = \mathbb{P}(\{\omega \in \mathcal{S}(\Theta_1)\}) = \mathbb{P}(\{\omega \in \mathcal{S}(\Theta_2)\}) = P(\Theta_2)$. An identical argument implies that $Q(\Theta_1) = \mathbb{Q}(\{\omega \in \mathcal{F}(\Theta_1)\}) = \mathbb{Q}(\{\omega \in \mathcal{F}(\Theta_2)\}) = Q(\Theta_2)$.

Moreover, as mentioned in Section 3.2.5, the allow groups $\{\omega_0^s, \omega_1^s, \dots\}$ and deny groups $\{\omega_0^f, \omega_1^f, \dots\}$ are simply requirements that certain $\omega \in 2^{\mathcal{T}}$ be elements of $\mathcal{S}(\Theta)$, and certain $\omega \in 2^{\mathcal{T}}$ be elements of $\mathcal{F}(\Theta)$, re-

spectively. Again, since $\mathcal{S}(\Theta)$ and $\mathcal{F}(\Theta)$ are preserved through logical transformations of the access formula, we conclude that the satisfaction of allow and deny group constraints is logically invariant. Finally, if $\Theta_1 \approx \Theta_2$ and $M(\Theta_1) \leq M_0$, then $M(\Theta_2) \leq M_0$ since the logical transformation from F_{Θ_1} to F_{Θ_2} would not introduce any new literals.

To summarize, the objective function and the first four constraints in (1) are all logically invariant. So, our strategy for finding approximate solutions to (1) will be to break the problem into two separate stages. In the first stage, we solve the following problem:

$$\begin{aligned} \min_{\Theta} \quad & Q(\Theta) \\ \text{s.t.} \quad & P(\Theta) \leq P_0 \\ & \{\omega_0^s, \omega_1^s, \dots\} \subseteq \mathcal{S}(\Theta) \\ & \{\omega_0^f, \omega_1^f, \dots\} \subseteq \mathcal{F}(\Theta) \\ & M(\Theta) \leq M_0 \end{aligned} \tag{2}$$

We denote by Θ^* the solution to the first-stage problem (2). In the second stage, we search for a configuration that is logically equivalent to Θ^*, which satisfies the remainder of the constraints in (1). The second-stage problem is, therefore:

$$\begin{aligned} \text{find} \quad & \Theta \\ \text{s.t.} \quad & \Theta \approx \Theta^* \\ & N(\Theta) \leq N_0 \\ & D(\Theta) \leq D_0 \\ & \mathcal{C}(\Theta) \in \mathcal{C}_0 \end{aligned} \tag{3}$$

Observe that the first-stage problem (2) involves an objective function and constraint set involving only logically invariant metrics. As such, given a solution Θ^* to (2), any configuration Θ' such that $\Theta' \approx \Theta^*$ will also solve (2). Therefore, without any loss of generality, we can assume that F_{Θ^*} is a formula in disjunctive normal form (DNF). This assumption reduces the complexity of problem (1) greatly since, in the first stage, we no longer have to worry about the different factorizations of a candidate solution. We only need to consider factorizations in the second-stage problem, by which point we have already found a Θ^*. As we will see, our two-stage strategy is a greedy approach to solving (1), so we are not guaranteed to find the globally optimum solution.

5 Solving the First-Stage Problem

To summarize, in the first stage we are trying to find a monotone DNF formula F_Θ that solves (2). However, we can equivalently search for a monotone truth table that solves (2), since any DNF formula (monotone or otherwise) can be recovered from its corresponding truth table. To recover the DNF formula, we simply form a single disjunction whose subclauses are the satisfying assignments (true rows) in the truth table, and then eliminate redundant subclauses. Of course, going in the other direction,

a	b	c	F
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Figure 6. Truth table for $F_\Theta = b + ac$.

we can always recover a truth table from any Boolean expression. For example, consider the truth table in Figure 6. The resulting disjunction is $\bar{a}b\bar{c} + \bar{a}bc + a\bar{b}c + ab\bar{c} + abc$. Simplifying this expression gives $F = b + ac$.

For the remainder of this paper, we adopt the following notation. The elements of $2^{\mathcal{T}}$ are labelled ω_i, where $i = 0 \dots 2^{M_0} - 1$, and the binary representation of i indicates the subset of $\mathcal{T}$ that ω_i represents. For example, with $M_0 = 4$, we have $\omega_{0110} = \{b, c\}$ and $\omega_{1111} = \{a, b, c, d\}$. The *complement* of ω_i is denoted by $\omega_{\overline{i}} \equiv \mathcal{T} \setminus \omega_i$. For example, again with $M_0 = 4$, we have $\omega_{\overline{0110}} = \omega_{1001} = \{a, d\}$ and $\omega_{\overline{1111}} = \omega_{0000} = \emptyset$. We then define $p_i \equiv \mathbb{P}(\omega_i)$ and $q_i \equiv \mathbb{Q}(\omega_i)$, respectively, as the probabilities of an attacker successfully breaking-into and destroying exactly the subset of terminals ω_i. Similarly, we write $p_{\overline{i}} \equiv \mathbb{P}(\omega_{\overline{i}})$ and $q_{\overline{i}} \equiv \mathbb{Q}(\omega_{\overline{i}})$.

For our purposes, a truth table is a list of 2^{M_0} Boolean values indicating whether each subset $\omega \in 2^{\mathcal{T}}$ is sufficient to reconstruct the root r. Formally, a *truth table* is defined as a function $T : 2^{\mathcal{T}} \to \{0, 1\}$ where $T(\omega_i) = 1$ if and only if $\omega_i \in \mathcal{S}(\Theta)$. We write $T_i \equiv T(\omega_i)$ and $T_{\overline{i}} \equiv T(\omega_{\overline{i}})$. Note that an equivalent definition of T would be: $T_i = 0$ if and only if $\omega_{\overline{i}} \in \mathcal{F}(\Theta)$. The intuition behind T is as follows. Suppose there is an input combination ω_i that causes F_Θ to evaluate to true (i.e., if $T_i = 1$). Then, an attacker can attempt to break-in to the set of terminals ω_i in order to reconstruct r. On the other hand, suppose the input combination ω_i causes F_Θ to be false, (i.e., if $T_i = 0$). Then, an attacker can instead attempt to destroy the terminals in $\omega_{\overline{i}}$, causing our sensitive data at r to be lost. The optimization problem (2), therefore, amounts to selecting the value of T_i, for every $\omega_i \in 2^{\mathcal{T}}$.

5.1 LP Formulation

We now show how to formulate (2) as a linear program (LP). An LP formulation allows us to solve the first-stage problem efficiently, using well-known techniques (e.g., the simplex algorithm).

Our first step is to cast (2) as an integer program (IP), as follows. First, observe that $P(\Theta) = \sum_{\omega_i \in \mathcal{S}(\Theta)} p_i = \sum_i p_i T_i$, since $T_i = 1$ for $\omega_i \in \mathcal{S}(\Theta)$ and $T_i = 0$ otherwise. Thus, we rewrite the first constraint in (2) as:

$$\sum_i p_i T_i \leq P_0 \tag{4}$$

Second, also observe that $Q(\Theta) = \sum_{\omega_i \in \mathcal{F}(\Theta)} q_i = \sum_i q_{\overline{i}}(1 - T_i)$, since $T_i = 0$ for $\omega_{\overline{i}} \in \mathcal{F}(\Theta)$ and $T_i = 1$ otherwise. The objective function in (2) can therefore be rewritten as:

$$\min_{\{T_i\}} \sum_i q_{\overline{i}}(1 - T_i) \tag{5}$$

The specifications of allow and deny groups are encoded as follows. We require that each allow group ω_i^s is a satisfying assignment. Thus, $T(\omega_i^s) = 1$. We require the each deny group ω_j^f is a falsifying assignment. Thus, $T(\mathcal{T} \setminus \omega_j^f) = 0$.

Finally, recall that T represents a monotone Boolean function. The implication of monotonicity is that, for every pair of sets (ω_i, ω_j) such that $\omega_i \subset \omega_j$, we have the constraint $T_i \leq T_j$. That is, if T is monotone and $\omega_i \subset \omega_j$, the only valid values for the ordered pair (T_i, T_j) are $(0,0)$, $(0,1)$ and $(1,1)$, but not $(1,0)$. The physical meaning of monotonicity is that by breaking into more terminals than he needs, an attacker does not somehow lose the ability to reconstruct the root r. Thus, the following IP results:

$$\begin{aligned}
\min_{\{T_i\}} \quad & \sum_i q_{\overline{i}}(1 - T_i) \\
\text{s.t.} \quad & \sum_i p_i T_i \leq P_0 \\
& T(\omega_i^s) = 1 \ \ \forall \omega_i^s \in \mathcal{S}(\Theta) \\
& T(\mathcal{T} \setminus \omega_j^f) = 0 \ \ \forall \omega_j^f \in \mathcal{F}(\Theta) \\
& T_i \leq T_j \ \ \forall (i,j) \text{ s.t. } \omega_i \subset \omega_j \\
& T_i \in \{0,1\} \ \ \forall i
\end{aligned} \tag{6}$$

Note that any solution T^* to the problem in (6) is exact. It will find a global optimum for the first-stage problem.

Recall that when we formulated (1), we assumed that $\mathbb{P}$ and $\mathbb{Q}$ were known probability measures. Knowledge of $\mathbb{P}$ and $\mathbb{Q}$ implies that the coefficients p_i and q_i in (6) are known constants. As such, the objective function and all constraints in (6) are, in fact, linear in the variables $\{T_i\}$. We can therefore form the *LP relaxation* of the IP (6) (see p.194 in [7]), by letting each T_i take on real values on the unit interval. The following LP results:

$$\begin{aligned}
\min_{\{T_i\}} \quad & \sum_i q_{\overline{i}}(1 - T_i) \\
\text{s.t.} \quad & \sum_i p_i T_i \leq P_0 \\
& T(\omega_i^s) = 1 \ \ \forall \omega_i^s \in \mathcal{S}(\Theta) \\
& T(\mathcal{T} \setminus \omega_j^f) = 0 \ \ \forall \omega_j^f \in \mathcal{F}(\Theta) \\
& T_i \leq T_j \ \ \forall (i,j) \text{ s.t. } \omega_i \subset \omega_j \\
& 0 \leq T_i \leq 1 \ , \ T_i \in \mathbb{R} \ \ \forall i
\end{aligned} \tag{7}$$

For moderate numbers of terminals, the LP (7) can be solved very efficiently. There is the issue that a feasible solution T' to (7) can be non-integral. That is, there may be some T_i' that is equal to neither 0 nor 1. The question is, how do we interpret such T_i'? A detailed treatment of this issue is beyond the scope of this paper, so we give only the important details here.

It can be shown that, for an optimum solution T^* of (7), most T_i^* values will in fact be either 0 or 1. These T_i^*'s typically account for the bulk of $\sum_i q_{\overline{i}}(1 - T_i)$ and $\sum_i p_i T_i$ at the optimum. Of the few T_i^*'s that are strictly between zero and one, some should be rounded up to one and the rest should be rounded down to zero. The key point is that it does not matter which specific T_i^*'s we choose to round up and which ones we round down. It only matters how many we round up and down. For example, suppose five T_i^* values are non-integral, and as many as two can be rounded up without violating the bound on $P(\Theta)$. Then, there are $\binom{5}{2} = 10$ possible combinations of T_i^*'s that can be rounded up, each of which corresponds to a different optimal solution of (7). As such, if more than one T_i^* turns out to be non-integral, then there are multiple optimum truth tables. Note that the set of optimum truth tables will be very "close" to each other, in that they will only differ by a small number T_i^*'s, since the vast majority of T_i^* values will be 0 or 1.

5.2 Graphical Interpretation

The search for a monotone Boolean formula has a nice graphical interpretation. Consider the M_0-dimensional hypercube, $[0,1]^{M_0} \subset \mathbb{R}^{M_0}$. We can represent each $\omega_i \in 2^{\mathcal{T}}$ as a vertex of this hypercube. For example, suppose $M_0 = 3$. Then ω_{111} (i.e., the set $\{a,b,c\}$) would map to the point $(1,1,1)$ and ω_{010} would map to $(0,1,0)$. We then associate the real numbers p_i and $q_{\overline{i}}$ with each vertex i of the hypercube.

In such a setting, it can be shown that any monotone truth table T can be represented by a *separating hyperplane* that cuts through the volume of this hypercube. This implication is a direct result of the monotonicity constraints in (7), namely $T_i \leq T_j \ \ \forall (i,j)$ s.t. $\omega_i \subset \omega_j$. A complete proof is given in [8], but Figure 7 conveys the important intuitions. The normal to the separating hyperplane lies in the non-negative orthant, $\mathbb{R}_+^{M_0}$. On one side of the hyperplane (the side further away from the origin) are ω_i for which $T_i = 1$. We call this the *true side* of the hyperplane. On the other side (the side closer to the origin) are the ω_i for which $T_i = 0$. We call this the *false side*. Thus, the truth table depicted in Figure 7 has $T_{11} = 1$ and $T_i = 0 \ \forall i \neq 11$.

Moreover, the points closest to the hyperplane on the true side are the *minterms* of T, and the points closest on the false side are the *maxterms* of T. For our purposes, we define a minterm of T to be an element $\omega_i \in 2^{\mathcal{T}}$ that is *minimally positive* i.e., $T_i = 1$ and $T_j = 0 \ \forall \ \omega_j \subset \omega_i$. We define a maxterm of T to be an element $\omega_i \in 2^{\mathcal{T}}$ that is *maximally negative* i.e., $T_i = 0$ and $T_j = 1 \ \forall \ \omega_j \supset \omega_i$. We are interested in minterms (maxterms) because the unique reduced DNF (CNF) representation of a monotone

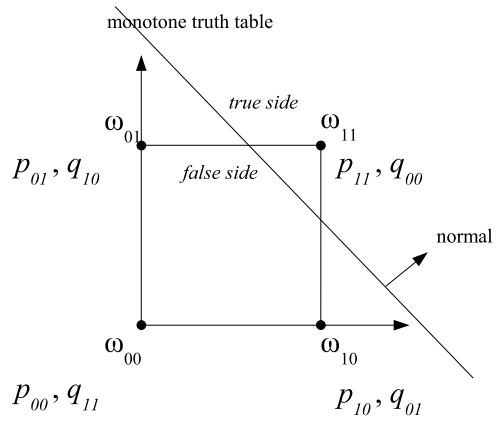

Figure 7. Example of a monotone truth table represented as a hyperplane.

truth table T is simply a disjunction (conjunction) of its minterms (maxterms). Thus, the minterms and maxterms are like "support vectors" for our separating hyperplane.

Using this graphical interpretation, the objective function $\sum_i q_{\overline{i}}(1-T_i)$ that we are trying to minimize is simply the sum of the $q_{\overline{i}}$ values for all points on the false side. The constraint $\sum_i p_i T_i \leq p_0$ is just an upper bound on the sum of the p_i values for all points on the true side. The allow groups are requirements that the $\{\omega_i^s\}$ lie on the true side, and the deny group constraints are requirements that the $\{\omega_i^f\}$ be on the false side. Finally, the upper bound on the number of terminals, M_0, determines the dimensionality of the hypercube we work in.

In summary, the problem of finding an optimal monotone truth table has the following graphical interpretation. We are searching for the separating hyperplane that minimizes the sum of $q_{\overline{i}}$ values for the hypercube vertices on the false side, while limiting the sum of the p_i values on the true side, and obeying the constraints that certain vertices lie on certain sides of the hyperplane. In our case, of course, the hypercube vertices correspond to elements of $2^{\mathcal{T}}$. The visualization in Figure 7 generalizes directly to higher dimensions.

5.3 Iterative Solution

The LP in (7) can be solved very efficiently for moderate numbers of terminal vertices, using widely available LP solvers. However, for larger sets of terminals, the number of constraints in (7) becomes large. In particular, for M_0 terminals, there are $M_0 2^{M_0-1}$ monotonicity constraints in 2^{M_0} variables, which, beyond roughly $M_0 = 20$, is too many constraints for even the most advanced LP solvers ($K2^{K-1}$ is the number of edges in a K-dimensional hypercube). As such, solving the LP becomes very difficult, even with the more advanced LP solvers.

In large problem instances, therefore, we must settle for an approximate solution to the first-stage problem. The graphical interpretation described in Section 5.2 leads us to a simple iterative algorithm for finding approximate solutions. The feature of the monotone truth tables that we rely on is that maxterms are the closest points to the hyperplane on the false side, and minterms are the closest on the true side. Formally, suppose T is a monotone truth table and let ω_i be one of the maxterms of T. By definition, $T_i = 0$. Let T' be another truth table identical to T, except that $T_i = 1$. Then, T' is also a monotone truth table, and ω_i is now a minterm of T'.

Suppose for simplicity that there were no allow or deny groups specified. The idea behind the iterative algorithm is that, starting with the all-zeros truth table, we make a sequence of small perturbations to the current truth table to arrive at a solution of (2). Each perturbation is simply selecting one of the maxterms of the current truth table and converting it to a minterm i.e., choosing one of the points closest to the hyperplane on the false side, and shifting the hyperplane slightly so that the point ends up on the true side. After each perturbation, $\sum_i p_i T_i$ is slightly higher and $\sum_i q_{\overline{i}}(1-T_i)$ is slightly lower, and most importantly, we still have a monotone truth table. We then continue perturbing until we can no longer do so without violating the constraint $\sum_i p_i T_i \leq P_0$.

A graphical intuition for the iterative procedure can be developed through an example in $\mathbb{R}^2$, as shown in Figure 8. The sequence of maxterms chosen is ω_{11}, ω_{10} and ω_{01}. The hyperplane (i.e., truth table) that results after step 3 represents the formula $F = a + b$.

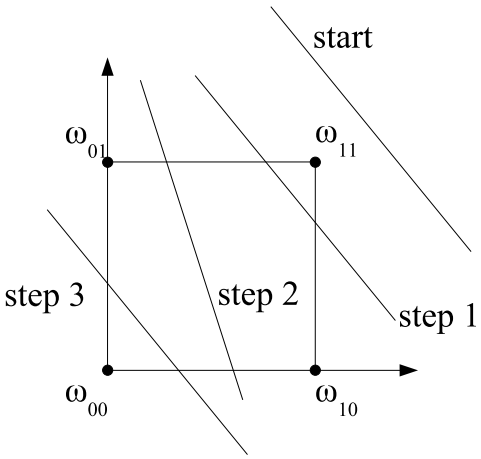

Figure 8. Example of iterative algorithm for finding the optimal monotone truth table.

Computing the maxterms of a monotone truth table T is just the *monotone CNF-DNF dualization* problem, which has been well studied (e.g., [9]). The only other issue to resolve, then, is how we select between the maxterms at each step. We must base our selection on a heuristic decision rule. For example, we might choose the maxterm with the largest $q_{\overline{i}}$ value (which corresponds to the largest gain in longevity), or instead the smallest p_i value (resp., smallest loss in privacy). We found, through our experiments, that the most effective heuristic was actually

Require: $P_0, M_0, \{\omega_i^s\}, \{\omega_i^f\}, \{p_i\}$ and $\{q_i\}$
$T \leftarrow$ all-zeroes truth table
for $i = 0$ to 2^{M_0-1} **do**
  if $\exists \omega \in \{\omega_0^s, \omega_1^s, \dots\}$ s.t. $\omega_i \subseteq \omega$ **then**
    $T_i \leftarrow 1$
  end if
end for
$P(\Theta) \leftarrow \sum_i p_i T_i$
while $P(\Theta) < P_0$ **do**
  $L \leftarrow maxterms(T) \setminus \{\omega_0^f, \omega_1^f, \dots\}$
  for all $\omega_i \in L$ **do**
    $z_i \leftarrow \frac{p_i}{q_i}$
  end for
  $L' \leftarrow L$ sorted by z_i value in descending order
  $k \leftarrow$ first element of L'
  $OK \leftarrow false$
  while $OK = false$ **do**
    if $P(\Theta) + p_k > P_0$, or setting $T_k \leftarrow 1$ would cause T to be unimplementable **then**
      if there are more elements in L' **then**
        $k \leftarrow$ next element of L'
      else
        return T
      end if
    else
      $OK \leftarrow true$
    end if
  end while
  $T_k \leftarrow 1$
  $P(\Theta) \leftarrow P(\Theta) + p_k$
end while
return T

Figure 9. Pseudocode for iterative algorithm for solving the first-stage problem (2).

the ratio of these two values. That is, we choose the maxterm of the current truth table that maximizes $\frac{q_i}{p_i}$. We just greedily select the highest gain in longevity at each step for the lowest amount of proportional privacy loss.

Now, if we had group constraints, we would only need to modify the iterative algorithm slightly. Rather than starting with $T_i = 0 \; \forall i$, we begin by setting $T(\omega) = 1$ for each allow group $\omega \in \{\omega_0^s, \omega_1^s \dots\}$. To ensure that we start with a monotone T, we also set $T(\omega) = 1$ for any $\omega \supset \omega_i^s$. Then, as the iterations progress, if we are ever presented with the option of adding a maxterm $\omega \in \{\omega_0^f, \omega_1^f \dots\}$, we do not do so (since we require $T(\omega_i^f) = 0$). We instead choose the next best maxterm instead (according to our heuristic).

Pseudocode for our iterative algorithm is given in Figure 9. The iterative algorithm is suboptimal because our choice of maxterm at each perturbation step is greedy. However, as we will show in our experiments, in certain circumstances our iterative algorithm actually finds the global optimum.

The iterative technique allows us to find a solution in $O(2^{M_0})$ iterations (one iteration per truth table entry), which is far better than solving an LP, which in our experiments was approximately quadratic in the number of constraints, $M_0 2^{M_0-1}$ [7]. As we discuss in Section 6, another benefit of the iterative technique is that we can efficiently handle the case where the second-stage problem is infeasible (i.e., there is no $\Theta \approx \Theta^*$ that meets all constraints in (3)). With the LP approach, on the other hand, we may need to solve a whole new LP.

6 Solving the Second-Stage Problem

To begin solving the second-stage problem in (3), we form a DNF expression F_Θ^* as a disjunction of the true rows of the first-stage solution T^*. The second-stage problem, then, is to find a configuration logically equivalent to Θ^* that satisfies the remainder of the constraints in (3). This corresponds to finding a good factorization of F_Θ^*.

A brute-force approach to finding a good factorization would be to exhaustively form all possible factorizations of F_Θ^*, and check if each factorization satisfies all of the constraints in (3). However, it is unclear how to exhaustively form all possible factorizations of F_Θ^*. Fortunately, there exist techniques in the digital circuit design literature (see [10] for a summary) that allow us to systematically explore a portion of the space of possible factorizations of F_Θ^* (though not the entire space). Thus, our approach will be to select a technique, and exhaustively check each element of the space explored by that technique.

Algebraic factoring considers a Boolean expression F to be a polynomial, and applies polynomial factorization laws to F. For example, the distributive law (i.e., $ab + ac = a(b + c)$) is within the scope of algebraic techniques. Very fast algorithms exist for algebraic factoring. *Boolean factoring* is a strictly more powerful technique that adds in the rules of Boolean logic e.g., $aa = a$ and $\bar{a} + a = 1$. Boolean factoring sometimes finds factorizations that algebraic factoring does not, since it uses a larger set of factoring rules. However, Boolean factoring is computationally very expensive. *Graph partitioning* [11] is a graph-theoretic technique that is not as expensive as Boolean factoring, but yields a larger set of factorizations than algebraic factoring.

For any given Boolean expression F, its *dual* F' is formed by replacing all conjunctions with disjunctions, and vice versa (note that $(F')' = F$). In many cases, factoring F' and dualizing the result yields factorizations that we would not have found by factoring F directly. Moreover, in our problem, we have the advantage that F_Θ^* is monotone. As such, some of the extra factoring rules allowed by Boolean factoring simply do not apply e.g., $a + \bar{a} = 1$. As such, the set of factorizations produced by algebraic factoring, Boolean factoring and graph partitioning will be very similar. Therefore, our strategy for solving (3) will be to apply algebraic factoring to both F_Θ^* and its dual, and check all resulting factorizations for fea-

sibility.

If we conclude that there is no factorization of F^*_Θ that is feasible, we must "back-off" from our first-stage solution, and find a new F^*_Θ. There are multiple ways to find a new F^*_Θ. Each method amounts to finding the next-best T^* from the first-stage solution. For example, if the LP solver we are using outputs a sequence of truth tables which were considered as the solver converged upon T^*, we could use the next-to-last one in the sequence. Similarly, if we used the iterative technique, we can undo our final step and use the maxterm with the second best $\frac{q_i}{p_i}$ ratio instead. Alternatively, we could modify one of the first-stage constraints, and re-solve the LP, to get a whole new T^*. Whichever method we use to "back-off", we use a new T^* and retry the second-stage problem.

In our experiments, we found that whenever the first-stage solution Θ^* was unimplementable, there would not be any logically equivalent configurations $\Theta' \approx \Theta$ such that Θ' was implementable (i.e., the second-stage was always infeasible). As such, we modified the iterative technique if Figure 9 (i.e., we added an extra condition to the *if*-clause in line 17) such that if adding a maxterm ever caused the configuration to become unimplementable, we would simply choose the next best maxterm instead. In the LP technique, we would simply lower the value of P_0, and re-solve the LP. There do exist DNF formulas F_Θ where Θ is unimplementable, but an implementable factorization exists. For example, $(a+c)(a+d)(a+e)(b+c)(b+d)(b+e)$ is unimplementable, but is logically equivalent to $(a+b)(c+d+e)$, which is implementable (in fact, it is read-once). However, in our experiments, such configurations never arose as solutions to our optimization problem.

7 Experimental Results

A few questions arise naturally from our discussion thus far, which are best answered through experiments. Firstly, how does the performance of the various first-stage solution techniques compare? Shown in Figures 10 and 11 is the probability of data loss, $Q(\Theta)$, for the solution Θ to (1) that is output by the LP and the iterative techniques, for various values of the bound on break-in probability, P_0. The $Q(\Theta)$ values for the global optimum, found through a brute-force search, are also shown. Four terminals were used, (i.e., $M_0 = 4$) and failures (i.e., break-ins and data loss) at each terminal were assumed to occur independently of all other terminals.

In Figure 10, we assume that each terminal is broken-into by an attacker with probability $P^t = 20\%$, and the data at each terminal is destroyed with probability $Q^t = 20\%$. The $Q(\Theta)$ achieved by the LP and iterative techniques are both equal to the global optimum, in this case (the three curves overlap). This result was found to apply even when the values of P^t and Q^t were varied. In Figure 11, the independent failures assumption was retained, but the terminals' failure characteristics were heterogeneous. We used a "strong" terminal ($P^t_a = 10\%$ and $Q^t_a = 10\%$), a "weak" terminal ($P^t_b = 20\%$ and $Q^t_b = 20\%$), one that

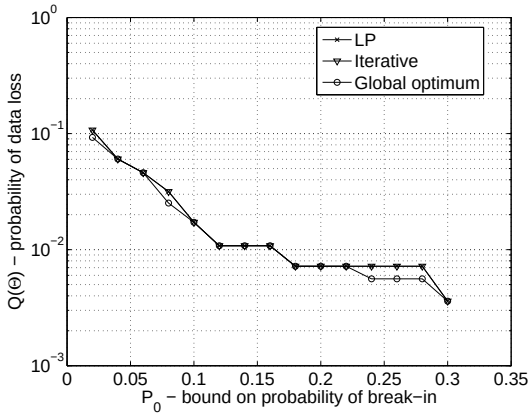

Figure 10. Comparison of first-stage solution techniques, using four identical terminals whose failures are independent.

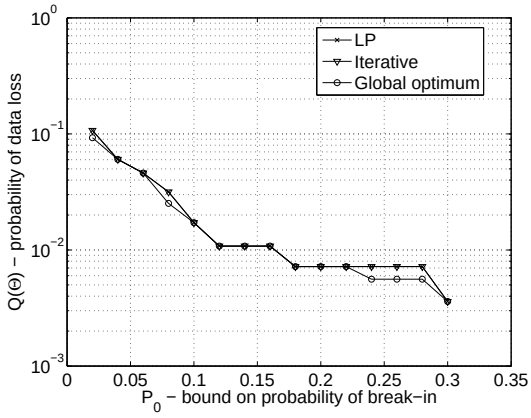

Figure 11. Comparison of first-stage solution techniques, using four heterogeneous terminals whose failures are independent.

was strong against break-ins but weak against data loss ($P^t_c = 10\%$ and $Q^t_c = 20\%$) and one that was the opposite ($P^t_d = 20\%$ and $Q^t_d = 10\%$). We see in this case that the outputs of the LP and iterative techniques are identical, but are both slightly worse than the global optimum for some P_0 values. Similar results were observed with various settings of terminal failure probabilities.

The performance of the LP and iterative techniques differ when correlations are introduced between the failures of terminals. Figure 12 shows the results using $M_0 = 6$ terminals, $P^t = 20\%$ and $Q^t = 20\%$ for all terminals, and the following two assumptions of correlated failure: when a is broken-into, so is b, and data loss occurs simultaneously at c and d. In this case, the $Q(\Theta)$ achieved by the LP is better than the iterative for most P_0 values. Results for the global optimum were not computed, since even with $M_0 = 6$, a brute-force search was intractable.

There is a caveat to the experimental results thus far. The configurations output by our first-stage solution techniques were often not implementable. So, we ask, what is

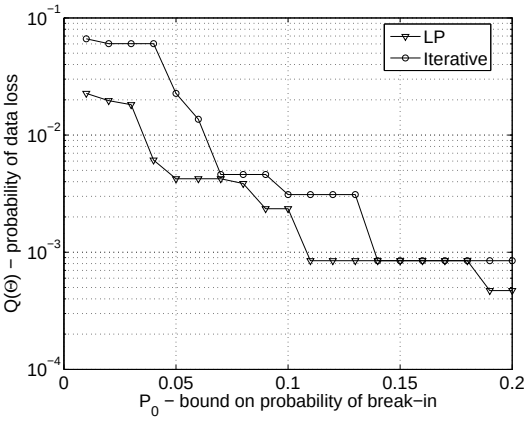

Figure 12. Comparison of first-stage solution techniques, using six identical terminals whose failures are correlated.

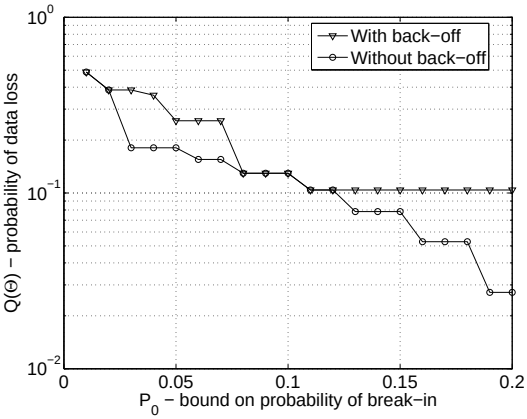

Figure 13. Comparison of $P(\Theta)$ achieved with four terminals, when implementability is considered.

the impact of requiring Θ to be implementable? Figure 13 compares the $Q(\Theta)$ values achieved by the iterative technique when we are required to "back off" from unimplementable solutions as described in Section 6, to the values achieved when we do not "back off". Here we assumed $M_0 = 4$ and $P^t = Q^t = 20\%$ – recall from Figure 10 that in this case the iterative technique found the global optimum. We see that for some P_0 values, we pay a large $Q(\Theta)$ penalty for requiring implementability. However, at the points where the two curves meet, the solution output by the iterative technique is actually implementable. From these results we see that implementability really is a significant issue – if we ignore it, we are likely to end up with an unimplementable configuration, whereas if we consider it, we often get reduced levels of protection.

In every experiment, without exception, whenever the first-stage solution was unimplementable, we found no logically equivalent configurations that were implementable. This observation was the justification, in the iterative technique (see pseudocode in Figure 9), for rejecting first-stage solutions that were unimplementable, without ever proceeding to the second-stage. Note that rejecting solutions in the first-stage saves us a lot of computational effort, both in exploring the space of factorizations in the second-stage, and checking the implementability of each factorization (which is a costly procedure). We do not incur the second-stage expense, only to be forced to return to the first-stage.

F_Θ	$D(\Theta)$	$N(\Theta)$	$\mathcal{C}(\Theta)$
$abc + bd + cd$	2	4	P
$(b + cd)(d + ac)$	3	5	P
$(c + bd)(d + ab)$	3	5	P
$b(ac + d) + cd$	4	5	P
$c(ab + d) + bd$	4	5	P
$abc + d(b + c)$	3	4	P
$(abc + d)(b + c)$	3	4	I
$(a + d)(b + c)(b + d)(c + d)$	2	5	I
$b(a + d)(c + d) + cd$	3	5	U
$c(a + d)(b + d) + bd$	3	5	U
$(b + cd)(a + d)(c + d)$	3	5	U
$(d + ac)(b + c)(b + d)$	3	5	U
$(c + bd)(d + a)(d + b)$	3	5	U
$(c + b)(c + d)(d + ab)$	3	5	U

Figure 14. Results of algebraically factoring $F_\Theta = abc + bd + cd$ and its dual. In the fourth column, P = proper, I = implementable and U = unimplementable.

A third question is concerns the space of factorizations explored via algebraic factoring of F_Θ and its dual F'_Θ. Are there enough factorizations? We consider as an example the configuration $F_\Theta = abc + bd + cd$, which is the first-stage solution when $M_0 = 4$, $P^t = Q^t = 20\%$ independently and $P_0 = 10\%$ (same parameters as Figure 10). F_Θ is implementable. Figure 14 lists configurations logically equivalent to Θ that were found using algebraic factoring, along with their depth $D(\Theta)$, number of nonterminals $N(\Theta)$ and class $\mathcal{C}(\Theta)$. We see that even for a relatively small number of terminals, there is a fairly rich set of alternative factorizations for us to choose from.

Fourthly, the scalability of each first-stage technique is of interest. As we mentioned, a brute-force search for a global optimum is intractable beyond roughly $M_0 = 5$ terminals (with $M_0 = 6$, there are more than 8 million monotone truth tables to check). In our experiments, using a PC with a 1.4 GHz processor and 1 GB of memory, we were able to run the LP technique for up to $M_0 = 12$ terminals, and the iterative technique for up to $M_0 = 20$ terminals, using "un-optimized" code in both cases. We conclude that our techniques can be used for the approximate solution of (1), with a reasonably large set of terminals. Observe that we did not specify allow or deny groups in our experiments – if we did, the problem size could be reduced greatly since many T_i values would be determined by these group constraints. Moreover, in the special case of identical terminals that fail independently,

there are optimizations that can be made which would allow us to compute solution for much larger problem instances.

Finally, we ask whether our techniques are actually helping us. That is, how useful are the techniques that we have developed compared to a "naive" strategy where terminals are conncected in a simple structure? Given M_0 terminals, a reasonable strategy might be to make $\frac{M_0}{2}$ copies of the root data r (assuming M_0 is even), and split each copy into two pieces. It is easy to show that, with $P^t = Q^t = 20\%$ and independent terminal failures, this strategy achieves $Q(\Theta) = (0.36)^{\frac{M_0}{2}}$ and $P(\Theta) = 1 - (0.96)^{\frac{M_0}{2}}$. Figure 15 compares these values to the $Q(\Theta)$ and $P(\Theta)$ at the optimum found by the iterative technique, for P_0 fixed at 20% and M_0 ranging between 4 and 12. We see that the probability of data loss, $Q(\Theta)$, achieved by the iterative technique is consistently an order of magnitude smaller than the naive strategy, and that this gap is gradually increasing with more terminals. Moreover, beyond roughly 11 terminals, the naive technique violates our constraint that $P(\Theta) < P_0 = 20\%$, whereas the iterative technique consistently makes efficient use of our specified tolerance for break-in probability, $P(\Theta)$. These results suggest that our techniques are, indeed, offering significant advantages over a seemingly reasonable (but naive) strategy.

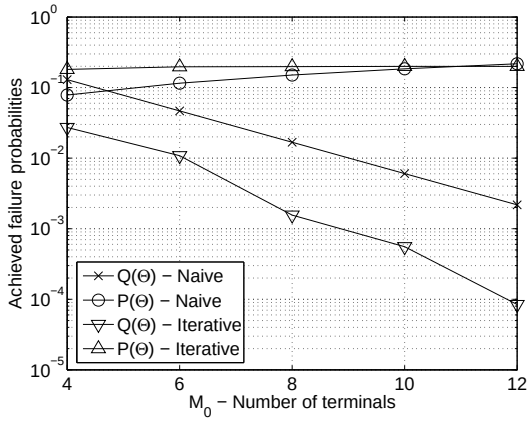

Figure 15. Comparison of $P(\Theta)$ achieved by the iterative algorithm and a naive strategy, using four terminals that fail independently.

8 Discussion

8.1 Dual Problem

All of the reasoning developed in this paper would apply directly to the "dual" problem of minimizing $P(\Theta)$ subject to an upper bound on $Q(\Theta)$, with only minor modifications. For example, in the LP (7), we interchange the inequality constraint and the objective function. In the iterative technique, we start instead with the all-ones truth table, and sequentially shift the hyperplane so that minterms are converted to maxterms. There are other minor changes, but all of the physical intuitions follow through completely.

8.2 A Generalization

Thus far, we have taken $(\Omega_P, \mathbb{P})$ and $(\Omega_Q, \mathbb{Q})$ to be independent probability spaces – terminals being broken-into do not statistically affect data being lost. However, it is straightforward to generalize our model and techniques to the correlated case, by considering a single new probability space $(\Omega_S, \mathbb{S})$, where $\Omega_S = \Omega_P \times \Omega_Q$ is a cross-product sample space, and $\mathbb{S}$ is a new failure distribution that accounts for arbitrary correlations between terminal data loss and break-ins. With this model, we could model a very malicious attacker, indeed e.g., one who destroys every terminal that he breaks-into. The resulting changes to our solution techniques are totally straightforward, although we do not discuss them here.

8.3 Other Related Work

The taxonomy of configurations (see Section 3.2.2) and related membership algorithms are developed in [5]. To our knowledge, the only other work that simultaneously addresses data privacy and longevity issues is [4]. They make use of threshold security schemes [12, 13], which split data into k shares – any $q < k$ shares can reconstruct the data, and fewer than q leak no information. Our Split operator uses $q = 1$, and our Copy uses $q = k$. However, configurations are compositions of simple primitives and allow asymmetry (e.g., Figure 1), whereas a threshold operator is symmetric and behaves as a single "black box".

Other relevant work [14, 15, 16] considers privacy issues when data is released for processing. They assume that data stored within the database is safe from both break-ins and data loss. Similarly, there is much work on data longevity (e.g., [17, 18]) which focuses on replication as a means of ensuring longevity.

9 Conclusions

Using our model of configurations, we defined metrics over the space of systems that safeguard data, and presented an optimization problem whose solution is an optimal configuration under specified tolerances and constraints. We have also presented a tractable solution methodology for this problem. Experimental results suggest that our techniques can yield significantly better solutions than a naive strategy. Our results also highlight the importance of properly dealing with unimplementable configurations.

Acknowledgement: We gratefully acknowledge Siddharth Joshi (sidj@stanford.edu) for his input during the development of our LP-based solution technique.

References

[1] M. L. Shooman, *Reliability of Computer Systems and Networks: Fault Tolerance, Analysis, Design*. New York, NY, USA: John Wiley and Sons, 2002.

[2] A. J. Menezes, S. A. Vanstone, and P. C. V. Oorschot, *Handbook of Applied Cryptography*. Boca Raton, FL, USA: CRC Press, 2001.

[3] E. Goh, H. Shacham, N. Modadugu, and D. Boneh, "Sirius: Securing remote untrusted storage," 2003.

[4] J. Wylie, M. Bigrigg, J. Strunk, G. Ganger, H. Kiliccote, and P. Khosla, "Survivable information storage systems," *IEEE Computer*, vol. 33, no. 8, pp. 61–68, Aug. 2000.

[5] B. Mungamuru, H. Garcia-Molina, and C. Olston, "Security configuration management," *Stanford InfoLab Technical Report*, 2005, http://dbpubs.stanford.edu/pub/2005-41.

[6] E. W. Weisstein, "Dedekind's problem," *MathWorld–A Wolfram Web Resource*, 1999.

[7] S. Boyd and L. Vandenberghe, *Convex Optimization*. New York, USA: Cambridge University Press, 2004.

[8] B. Mungamuru, H. Garcia-Molina, and S. Mitra, "How to safeguard your sensitive data," *Stanford InfoLab Technical Report*, 2006, http://dbpubs.stanford.edu/pub/2006-9.

[9] T. Eiter, G. Gottlob, and K. Makino, "New results on monotone dualization and generating hypergraph transversals," in *34th Symposium on Theory of Computing*, 2002.

[10] G. deMicheli, *Synthesis and Optimization of Digital Circuits*. New York, USA: McGraw-Hill, 1994.

[11] A. Mintz and M. C. Golumbic, "Factoring boolean functions using graph partitioning," *Discrete Appl. Math.*, vol. 149, no. 1-3, pp. 131–153, 2005.

[12] G. R. Blakley and C. Meadows, "Security of ramp schemes," in *Proceedings of the CRYPTO '84 Conference on Advances in Cryptology*, 1984.

[13] A. Shamir, "How to share a secret," *Communications of the ACM*, vol. 22, no. 11, pp. 612–613, Nov. 1979.

[14] R. Agrawal, J. Kiernan, R. Srikant, and Y. Xu, "Hippocratic databases," in *Proc. of the 28th Int'l Conference on Very Large Databases*, 2002.

[15] V. S. Verykios, E. Bertino, I. N. Fovino, L. P. Provenza, Y. Saygin, and Y. Theodoridis., "State-of-the-art in privacy preserving data mining," *SIGMOD Record*, vol. 33, no. 1, pp. 50–57, 2004.

[16] L. Sweeney, "k-anonymity: a model for protecting privacy," *International Journal on Uncertainty, Fuzziness and Knowledge-based Systems*, vol. 10, no. 5, pp. 557–570, 2002.

[17] B. Cooper and H. Garcia-Molina, "Peer-to-peer data trading to preserve information," *ACM Transactions on Information Systems*, vol. 20, no. 2, pp. 133–170, Apr. 2002.

[18] V. Reich and D. S. H. Rosenthal, "LOCKSS: Lots of copies keeps stuff safe," in *Proceedings of the 2000 Preservation Conference*, Dec. 2000.