# Relating Boolean Gate Truth Tables to One-Way Functions

Mahadevan Gomathisankaran* Akhilesh Tyagi†

March 3, 2008

## Abstract

We present a schema to build one way functions from a family of Boolean gates. Moreover, we relate characteristics of these Boolean gate truth tables to properties of the derived one-way functions. We believe this to be the first attempt at establishing cryptographic properties from the Boolean cube spaces of the component gates. This schema is then used to build a family of compression functions, which in turn can be used to get block encryption and hash functions. These functions are based on reconfigurable gates. We prove cryptographically relevant properties for these function implementations. Various applications incorporating these one-way functions, specifically memory integrity in processor architecture, are presented.

## 1 Introduction

One way functions are at the heart of cryptography. In this paper, we explore a Boolean cube space schema and a hardware implementation for a preimage-resistant function, which we call one-way function, within the bounds of acceptable notational abuse. The primary motivation for our research arose in the realm of secure processor architecture. One of the desirable properties for a secure execution environment is data or memory integrity. Simply stated, memory integrity implies that a memory read from address $A$ at time $T$, $M[R,A,T]$, must return the latest written value into address $A$, $M[R,A,T] = V$ iff $\exists M[W,A,V,T']$ for $T' < T$ and $\forall t \in [T'+1, T-1]$, $\nexists M[W,A,*,t]$. Here $M[R,A,T$ is a memory read from address $A$ at time $T$, and $M[W,A,V,T]$ is a memory write of value $V$ at address $A$ at time $T$. The trust boundary is assumed to be within the processor, the memory is untrusted. Aegis [8] provides such memory integrity by maintaining a Merkle hash tree [5] for the entire protected address space. The root of the hash tree is maintained within the trusted processor boundaries, but the rest of the tree nodes can be kept in the untrusted memory. Suh et al. [8] use a general hash function such as SHA-1 with the assumed latency of 160 cycles. A general hash function has to provide collision resistance when the adversary has access to the hash function oracle. We note however that in the memory integrity scenario, the adversary only gets to see the hashed value $H_K(V)$. It does not have access to a hash oracle, wherein, $H_K$ function can be exercised and observed at many different domain points. The secure processor environment prevents adversary access to the plain-text input. The hashed digests $H_K(V)$ resident in untrusted memory are however observable. In such a scenario, the property of interest is really preimage or inversion resistance. This research was motivated by this scenario in order to provide an efficient, low-latency implementation of an inversion resistant function.

The design objective for an inversion resistant function $f : \{0,1\}^m \rightarrow \{0,1\}^n$ for $m > n$ is to distribute the preimages $f^{-1}(y)$ as uniformly as possible. We state this objective in terms of two properties of the truth tables for individual output bits of $y$, $y_{n-1}, y_{n-2}, \ldots, y_0$. One of these properties ensures equal frequency for 0s and 1s in a variable, and the other one guarantees equal frequency for all the four outcomes 00, 01, 10, 11 when two variables are instantiated simultaneously. The first property is *balance* or lack of bias in 0s and 1s, and the second property is *independence*. Balance and independence together maintain uniform distribution for all the output bit sequences $y \int d\{0,1\}^n$. Independence also assures that even if a small number of input-output relationships $(x,y) \mid f(x) = y$ are exposed to an adversary, the damage is limited to only these instances. It does not leak any information about other input-output relationships. We then explore the properties required of smaller, physically realizable Boolean gates (of fanin of the order of 4) that lead to balance and independence in the output bits $y_i$'s. These Boolean gates are also called balanced or unbiased and independent. Finally, we develop a composition mechanism for these unbiased and independent gates that propagates these properties into larger functions. We describe these gates and composition in Sections 2 and 3. Analysis of the preimage resistance for this schema is also presented in Section 3. The memory integrity application and other uses for this one way function are described in

*Department of Electrical Engineering, Princeton University, Princeton, NJ 08536. email:mgomathi@princeton.edu

†Department of Electrical and Computer Engineering, Iowa State University, Ames, IA 50011. email:tyagi@iastate.edu

Section 4. We conclude the paper in Section 5.

## 2 Building Blocks: Leaf Gates & Switchbox Gates

We describe the building blocks of the proposed one way functions in this section. A one way function will be composed from many fixed-arity gates with a set of prespecified properties. We call these functions *leaf gates*. We first define two properties of Boolean functions that are of interest to us.

**Definition 2.1** (Unbiased Function)**.** *A Boolean function* $f : \{0,1\}^n \to \{0,1\}$ *is said to be* **unbiased** *iff* $P(f=0) = P(f=1)$. *We will use* balanced *and* unbiased *interchangeably.*

**Definition 2.2** (Independence)**.** *Let* $f_i, f_j : \{0,1\}^n \to \{0,1\}$ *be two functions with an identical support set.* $f_i$ *and* $f_j$ *are said to be* **independent** *iff* $P((f_i = x) \cap (f_j = y)) = P(f_i = x) \cdot P(f_j = y)$, *for* $x, y \in \{0,1\}$.

An unbiased function outputs both 0 and 1 with equal frequency over a uniformly distributed support set. Two independent functions, when randomly sampled, show all four output combinations $(0,0), (0,1), (1,0), (1,1)$ with the same likelihood (probability $1/4$ each). These two properties capture the *oblivious-ness* of gates which indicates their potential for information leak. In the following, $\Phi$ will denote a set of properties. These properties could be specified as predicates or some other form of a characteristic function. $\phi$ will refer to a specific property in $\Phi$. We use $\emptyset$ to denote an empty property which is satisfied by every function.

**Definition 2.3** ($n$-to-1 leaf gates)**.** *Let* $\mathcal{G}(n,\Phi)$ *be the set of all* **n-to-1** *Boolean functions from* $\{0,1\}^n \to \{0,1\}$ *that satisfy properties in the set* $\Phi$. *The two properties of interest to us are* unbias *denoted by* $\phi_{ub}$ *and* independence *denoted by* $\phi_{in}$. *We will write* $\mathcal{G}(n,\phi_{ub})$, $\mathcal{G}(n,\phi_{in})$, *and* $\mathcal{G}(n,\phi_{ub-in})$ *to indicate set of all n-input Boolean functions that are unbiased, independent, and unbiased & independent respectively.*

These leaf gates, specifically 4-to-1 leaf gates, will be used to build the larger one way functions. They would be wrapped into another switchbox as shown in Figure 1. The wrapper function performs an exclusive-or (xor) of the output from the leaf gate with another input, which we call *control* input. Depending on the value of the control input $x_n$, the output $y$ is either $f(n,\phi_{ub-in})$ or $\overline{f(n,\phi_{ub-in})}$. Note that $f(n,\Phi)$ denotes a specific gate from the family $\mathcal{G}(n,\Phi)$. A formal definition follows.

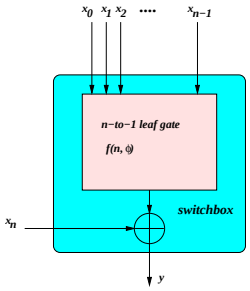


Figure 1: Switchbox Wrapper for $n$-to-1 functions

**Definition 2.4** ($n+1$-to-1 switchbox gates)**.** $\mathcal{S}(n+1, \mathcal{G}(n,\Phi), x_n)$ *is the set of* **(n+1)-to-1** *Boolean functions from* $\{0,1\}^{n+1} \to \{0,1\}$ *derived from* $\mathcal{G}(n,\Phi)$ *as follows. A function* $s(f, n+1) \in \mathcal{S}(n, \mathcal{G}(n,\Phi), x_n)$ *is constructed from* $f \in \mathcal{G}(n,\Phi)$ *and a control input* $x_n$ *as* $s(x_0, x_1, \ldots, x_{n-1}, x_n) = x_n \oplus f(x_0, x_1, \ldots, x_{n-1})$. *Note that we have made* $\mathcal{G}$ *and control input* $x_n$ *implicit in the definition of s in order to reduce the clutter in notation.*

We claim that the *switchbox* gates so defined are both unbiased and independent provided their inputs are unbiased and independent.

**Lemma 2.1** (unbiased switchbox gate)**.** *A switchbox gate* $s(f, n+1)$ *is unbiased if its inputs* $x_0, x_1, \ldots x_{n-1}, x_n$ *are unbiased.*

*Proof.* Note that we have not made any assumptions on the balance of the leaf gate $f(n,\Phi)$ that is embedded inside the given switchbox gate $s(f, n+1)$ (along the lines of Figure 1). In other words, the leaf gates come from the set $\mathcal{G}(n,\emptyset)$. We do however insist on balance (between 0s and 1s) in the inputs $x_0, x_1, \ldots x_{n-1}, x_n$.

Let $x_n$ be the control input for the switchbox $s$. Since $x_n$ is unbiased (balanced), the probabilities of $x_n = 0$ and $x_n = 1$ are equal, $P(x_n = 0) = P(x_n = 1) = 1/2$. Let the truth table for $f(n,\emptyset)$ be given by $T(f) = [f_0, f_1, \ldots, f_{2^n-1}]$, where $f$ evaluates to $f_j$ on input $[x_{n-1} \ldots x_0]$ equal to $j$. The truth table for $s$ (output $y$) is given by $[T(f)\ T(\overline{f})]$ taking $x_n$ to be the most significant bit. This is because the xor gate in $s$ acts as identity for $x_n = 0$ and as inverse (not) for $x_n = 1$. Let the number of 0's in $T(f)$ be $n_0(T(f))$ and the number of 1's be $n_1(T(F)) = 2^n - n_0(T(f))$. We also know the number of 0's and 1's in $T(\overline{f})$ to be $n_0(T(\overline{f})) = n_1(T(f))$ and $n_1(T(\overline{f})) = n_0(T(f))$. Hence $n_0(T(s)) = n_0(T(f)) + n_0(T(\overline{f})) = n_0(T(f)) + n_1(T(f))$ and $n_1(T(s)) = n_1(T(f)) + n_1(T(\overline{f})) = n_1(T(f)) + n_0(T(f))$. This implies that $n_0(T(s)) = n_1(T(s))$ leading to $P(y=0) = P(y=1) = 1/2$, which holds under the assumption that $y = f$ with probability 1/2 and $y = \overline{f}$ with probability 1/2 implied by the balance of $x_n$. □

**Lemma 2.2** (switchbox gate independence)**.** *Switchbox gates $s_0(f_0, n+1)$ and $s_1(f_1, n+1)$ are pairwise independent over the support set $\{x_0,\ x_1,\ \ldots x_{n-1}\}$, with unbiased and independent control inputs $x_n^0$ and $x_n^1$ respectively, assuming $f_0 \neq f_1$ and $f_0 \neq \overline{f_1}$.*

*Proof.* Note that for disjoint support sets for $s_0$ and $s_1$, the independence follows trivially. Hence, we consider the case where the support set $\{x_0, x_1, \ldots x_{n-1}\}$ is shared. The independence for $s_0$ and $s_1$ holds iff $f_0 \neq f_1$, $f_0 \neq \overline{f_1}$, and the control inputs $x_n^0$ and $x_n^1$ are unbiased and independent. Recall that independence of $s_0$ (output $y_0$) and $s_1$ (output $y_1$) is equivalent to showing that $P(y_0 = 0, y_1 = 0) = P(y_0 = 0, y_1 = 1) = P(y_0 = 1, y_1 = 0) = P(y_0 = 1, y_1 = 1) = 1/4$. If we consider the joint truth table of $(y_0, y_1)$, denoted by $T(s_0, s_1)$, assuming $x_n^0$ as the most significant bit and $x_n^1$ as the next significant bit, we get $T(s_0, s_1) = [T(f_0, f_1)\ T(f_0, \overline{f_1})\ T(\overline{f_0}, f_1)\ T(\overline{f_0}, \overline{f_1})]$. Let $n_{0,0}(T(f_0, f_1))$ denote the number of $(0,0)$ rows out of $2^n$ truth table entries. Similarly define $n_{0,1}(T(f_0, f_1))$, $n_{1,0}(T(f_0, f_1))$, and $n_{1,1}(T(f_0, f_1))$.

Next consider $n_{0,0}(T(f_0, \overline{f_1}))$, $n_{0,1}(T(f_0, \overline{f_1}))$, $n_{1,0}(T(f_0, \overline{f_1}))$, and $n_{1,1}(T(f_0, \overline{f_1}))$. Note that all instances of $(0,0)$ in $T(f_0, f_1)$ will be transformed into $(0,1)$ instances in $T(f_0, \overline{f_1})$. This implies that

$$\begin{aligned} n_{0,0}(T(s_0, s_1)) &= n_{0,0}(T(f_0, f_1)) + \\ &\quad n_{0,0}(T(\overline{f_0}, f_1)) + \\ &\quad n_{0,0}(T(f_0, \overline{f_1})) + \\ &\quad n_{0,0}(T(\overline{f_0}, \overline{f_1})) \end{aligned}$$

Moreover, this same argument also leads to

$$\begin{aligned} n_{0,1}(T(s_0, s_1)) &= n_{0,0}(T(f_0, f_1)) + \\ &\quad n_{0,1}(T(f_0, f_1)) + \\ &\quad n_{1,0}(T(f_0, f_1)) + \\ &\quad n_{1,1}(T(f_0, f_1)) \end{aligned}$$

In general, this shows that

$$\begin{aligned} n_{0,0}(T(s_0, s_1)) &= n_{0,1}(T(s_0, s_1)) \\ &= n_{1,0}(T(s_0, s_1)) \\ &= n_{1,1}(T(s_0, s_1)) \\ &= 1/4 \end{aligned}$$

This establishes the independence of switchbox functions. □

We will need another stronger notion of independence for the later proofs of compression function independence.

**Lemma 2.3** (switchbox gate independence under shared control)**.** *Switchbox gates $s_0(f_0, n+1)$ and $s_1(f_1, n+1)$ are pairwise independent over disjoint support sets $\{x_0^0, x_1^0, \ldots x_{n-1}^0\}$ and $\{x_0^1,\ x_1^1,\ \ldots x_{n-1}^1\}$ respectively, with shared control input $x_n$, assuming unbiased and independent $f_0$ and $f_1$.*

*Proof.* We need to prove that $P(s_0 \cap s_1) = P(s_0)P(s_1)$. Given disjoint support sets, the only intersection occurs with respect to $x_n$ For a fixed value of $x_n$ ($x_n = 0$ or $x_n = 1$), the functions $f_0$ and $f_1$ are equally likely to be 0 or 1 given their balance and independence. Hence for $x_n = 0$, $P(s_0 = 0, s_1 = 0) = P(s_0 = 0, s_1 = 1) = P(s_0 = 1, s_1 = 0) = P(s_0 = 1, s_1 = 1) = 1/4$. The same situation holds for $x_n = 1$. □

We will next show a constructive schema for the proposed family of one-way functions based on the switchbox gates.

# 3 One Way Function Schema

In this section, we will show a construction for an unbiased, independent Boolean function over 128 input bits from the switchbox gates $\mathcal{S}(5, \mathcal{G}(4, \phi_{ub}), x_4)$ (derived from the leaf gates $\mathcal{G}(4, \phi_{ub})$). This function will only use 85 of the 128 input bits.

**Definition 3.1** (scalable one-way function)**.** *A scalable one-way function $F : \{0,1\}^N \times \{0,1\}^{(N-1)/(k-1)} \rightarrow \{0,1\}$ is built from switchbox gates $\mathcal{S}(k+1, \mathcal{G}(k, \phi_{ub}), x_k)$ as follows. The primary inputs come from a set $I$ of cardinality $N$ and the control inputs come from a set $\mathcal{C}$ of cardinality $(N-1)/(k-1)$. $F$ is built as a tree of gates $s_{i,j}^F$ for $0 \leq i < log_k N$ and $0 \leq j \leq (N/k^{i+1})$ as shown in Figure 2. The level number within the tree is captured by i, and $j$ is the index within the level. The inputs to the $N/k$ gates at Level 0 come from the input set $I$. All the other inputs into the $f$ gates are internal. The control inputs come from an additional source of unbiased, independent variables $\mathcal{C}$. Let $\mathcal{F}(N, \mathcal{S})$ be the set of all scalable $N$-input functions realized from the switchbox gates $\mathcal{S}$. We will often omit $\mathcal{S}$ to denote this family by $\mathcal{F}(N)$.*

Now we construct a family of compression functions to compress a set of 128-bits into 64-bits built from scalable one-way functions from $\mathcal{F}(64)$.

**Definition 3.2** (compression function)**.** *$\mathcal{K}(N,k)$ is a family of compression functions. A compression function $\kappa : \{0,1\}^N \times \{0,1\}^N \rightarrow \{0,1\}^N \in \mathcal{K}(N,k)$ can be built with the scalable one-way functions from the family $\mathcal{F}(N, \mathcal{S})$ as follows. The set of $2N$ input bits is partitioned into $(I, \mathcal{C})$ of cardinality $N$ each. We select $N$ unique functions from $\mathcal{G}(k, \phi_{ub})$: $\{f_0, f_1, \ldots, f_{N-1}\}$ randomly. Let*

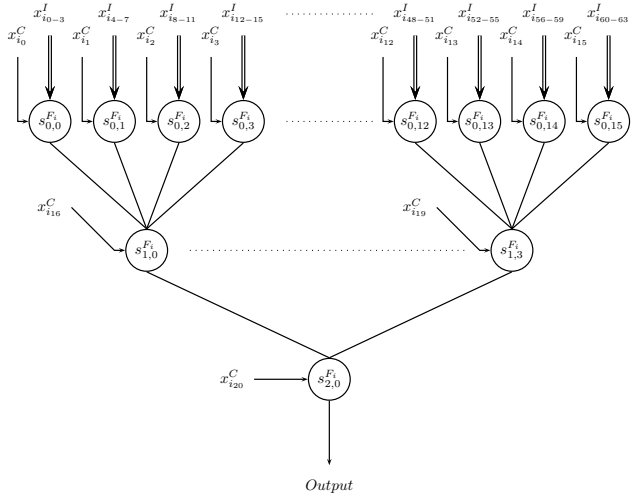


Figure 2: Construction of $F_i$ from switchgates over the input domain $(I, C)$

*$I = \{x_0^I, x_1^I, \ldots, x_{N-1}^I\}$ and $C = \{x_0^C, x_1^C, \ldots, x_{N-1}^C\}$. We construct $N$ unique, unbiased and independent, scalable one-way functions $F_i$ for $0 \leq i < N$. Each function $F_i$ contains $(N-1)/(k-1)$ leaf gates. Let $F_i^j$ be the $j$th gate of $F_i$ in row-major order for $0 \leq j < (N-1)/(k-1)$. $F_i^j$ corresponds to $s_{i',j'}^{F_i}$ from Definition 3.1 as follows: $j = \left(\frac{N-1}{k-1}\right) - \left(\frac{k^{log_k N - i' - 1} - 1}{k-1}\right) - \left(\frac{N}{k^{i'+1}} - j'\right)$. $F_i^j$ (as in Figure 2) is bound to $f_{(i+j)\%N}$. The control input for the gate $F_i^j$ is selected from $C$ as $x_{(i+2j)\%N}^C$. Let $C_i = \{x_l \mid l = (i+2j)\%N, 0 \leq j < (N-1)/(k-1)\}$ denote the set of control variables used in $F_i$. We note that $F_i : I \times C_i \rightarrow \{0,1\}$ for $0 \leq i < N$.*

This construction ensures that each of the $F_i$s is unbiased and independent.

### 3.1 Analysis

In this section, we establish that the compression functions from $2N$ to $N$ bits constructed with $k$-bit gates given by $\mathcal{K}(N,k)$ in Definition 3.2 are unbiased and independent.

**Lemma 3.1** (compression function balance)**.** *A compression function $\kappa \in \mathcal{K}(N,k)$ is* unbiased.

*Proof.* Assume that all the primary inputs $(I, C)$ are unbiased and independent. It suffices to show that each scalable function $F_i$ (from Definition 3.1) is unbiased. The proof then is by induction on the level number in the construction for $F$. The induction basis is for the output of Level 0 gates (whose $k$ inputs come from the set $I$). Lemma 2.1 establishes that these outputs are unbiased. Note that the Level 0 outputs feed Level 1 switchbox gates.

The inductive hypothesis is that the outputs of Level $l$, $l > 0$ switchbox gates are unbiased. In order to establish that the outputs of Level $(l+1)$ gates are unbiased, we observe that the inputs to Level $(l+1)$ gates are unbiased. Lemma 2.1 argues as a subcase that for unbiased inputs, the output of a switchbox gate is also unbiased. Note that the control input $x_n^C$ from $C$ is unbiased. Recall that the truth table for $s$ is given by $[T(f)\ T(\overline{f})]$. A simple counting argument establishes that $P(y=0) = P(y=1) = 1/2$ only assuming that $P(x_n = 0) = P(x_n = 1) = 1/2$. □

We prove two notions of independence for the compression function output bits $F_i$. The black-box independence assumes that the adversary has access to only the output bits $F_i$s, and hence independence only with respect to the event $F_i \cap F_j$ is required. On the other hand, if the adversary can have white-box access to the compression function, wherein, all the internal gates are also observable, independence will have to be established with respect to the event $F_i^p \cap F_j^q$ for arbitrary internal gates $0 \leq p, q < (N-1)/(k-1)$.

**Lemma 3.2** (compression function independence)**.** *A compression function $\kappa \in \mathcal{K}(N,k)$ is both blackbox and white-box* independent.

*Proof.* We need to show that $F_i$ and $F_j$ are pairwise independent for $0 \leq i \neq j < N$. The proof will again be by induction. The induction basis is that the corresponding Level 0 outputs, $s_{0,j'}^{F_i}$ and $s_{0,j'}^{F_j}$ for $0 \leq j' < N/k$, are pairwise independent, which follows from Lemmas 2.2 and 2.3. For Lemma 2.2, we need the following two assumptions. The two assumptions in Lemma 2.2 are $f_{0,j'}^{F_i} \neq f_{0,j'}^{F_j}$ (or $f_{0,j'}^{F_i} \neq \overline{f_{0,j'}^{F_j}}$) and independence of $x_{0,j'}^{F_i C}$ and $x_{0,j'}^{F_j C}$. In other words, we need to show that these assumptions hold for all gates $F_i^p$ and $F_j^p$ for $0 \leq p < (N-1)/(k-1)$. Recall from Definition 3.2 that Gate $F_i^p$ ($F_j^p$) is bound to the truth table $f_{(i+p)\%N}$ ($f_{(j+p)\%N}$) respectively. Note that $(i+p)\%N \neq (j+p)\%N$ for $i \neq j$. Similarly, the control input for $F_i^p$ ($F_j^p$) is $x_{(i+2p)\%N}^C$ ($x_{(j+2p)\%N}^C$) respectively, which are different for $i \neq j$. The inductive hypothesis is that $s_{i',j'}^{F_i}$ and $s_{i',j'}^{F_j}$ are pairwise independent for $0 \leq i' \leq l$. This has established only that each pair $F_i^p$ and $F_j^p$ are independent. In order to ascertain that the outputs across a level are independent, we also need Lemma 2.3 to cover the case when the two gates $F_i^p$ and $F_j^q$ share the same control input $x_n$ for $p \neq q$ and $i \neq j$. Note that this is the reason why in Definition 3.2 we insisted on unbiased leaf gates $f$. The mapping for truth tables $f_i$s and control inputs $x_i^C$s in Definition3.2 ensures that an arbitrary pair of gates $F_i^p$ and $F_j^q$ are disjoint in at least two of the following three parameters: sup-

port set, control input, and truth tables. Hence, Lemmas 2.2 and 2.3 together cover the universe.

The inductive step again follows from Lemmas 2.2 and 2.3 since the independence of the inputs to Level $l+1$ is established by the inductive hypothesis. □

We now need to prove preimage resistance for the compression functions $\kappa$ in $\mathcal{K}(N,k)$. An intuitive estimate of number of $\kappa$-oracle queries needed of an adversary to determine an element $x \in \{0,1\}^{2N}$ in the preimage of a randomly selected $y \int d\{0,1\}^N$, $\kappa(x) = y$ is $2^{2N}/2^N = 2^N$. We adopt the Shannon blackbox model, specifically, Black, Rogaway, Shrimpton version [1] for this purpose. We quantify the adversary advantage over $q$ oracle queries. We will use the function $\mathbf{Adv}_f^{Pre}(A)$ from Rogway, Shrimpton [7]. We will use $x \xleftarrow{\$} S$ to denote a random selection of $x$ from set $S$. The definition we use is as follows.

**Definition 3.3** (preimage resistance)**.** $\mathbf{Adv}_\kappa^{Pre}(A) = P(I \xleftarrow{\$} \{0,1\}^N;\ C \xleftarrow{\$} \{0,1\}^N;\ \kappa \xleftarrow{\$} \mathcal{K}(N,k);\ \sigma \leftarrow \kappa(I,C);\ (I',C') \xleftarrow{\$} A(\sigma) \mid \kappa(I',C') = \sigma)$.

The one-way function $\kappa \in \mathcal{K}(N,k)$ is chosen. Specifically, the truth tables of all the gates ($f_i$s) constitute the secret. We will allow the adversary to pick two $N$ bit inputs $I \in I$ and $C \in \mathcal{C}$. The oracle returns a $Y \in \{0,1\}^N$ such that $Y = \kappa(I,C)$. We repeat this experiment $q \leq 2^{N-1}$ times, and quantify the information the adversary collects over these repeated experiments.

**Theorem 3.1** (preimage resistance)**.** *Fix $N$, $k$, and $\kappa \in \mathcal{K}(N,k)$.* $\mathbf{Adv}_\kappa^{Pre}(q) \leq q/2^{N-1}$ *for any $q \geq 1$.*

*Proof.* Once again, along the lines of Black et al. [1], let $A^?$ be an adversary, where ? denotes a query to the $\kappa$ oracle. $A$ is allowed exactly $q$ such queries. A's behavior is identical to the following. Initially, $i \leftarrow 0$ and $\kappa(I,C) =$ *undefined* for all $(I,C) \in \{0,1\}^N \times \{0,1\}^N$. Let $A^?$ be run as follows. On a query $(I,C)$, $i \leftarrow i+1$; $I_i \leftarrow I$; $C_i \leftarrow C$; $Y_i \xleftarrow{\$} \overline{Range}(\kappa)$; $\kappa(I,C) \leftarrow Y_i$; return $Y_i$ to $A$. $Range(\kappa)$ is the set where $\kappa(I,C)$ is no longer *undefined*, and $\overline{Range}(\kappa) = \{0,1\}^N - Range(\kappa)$. Note that the implicit assumption here that learning one association $(I_i,C_i,Y_i)$ does not help predict any other future associations for $\kappa$ is grounded in independence argument from Lemma 3.2. When $A$ halts outputting *out*, the oracle simulator program returns $((I_1,C_1,Y_1),\ldots,(I_q,C_q,Y_q),out)$ along the lines of [1]. If $A$ succeeds, it outputs $(I_i,C_i,Y_i)$, $1 \leq i \leq q$ such that $\kappa(I_i,C_i) = Y_i = \sigma$. Let $S_i$ be the event that $(I_i,C_i,Y_i)$ satisfies $\kappa(I_i,C_i) = Y_i = \sigma$. However, the simulation of $A$'s oracle assigns $Y_i$ randomly from a set of size at least $2^N - (i-1)$. Hence $P(S_i) \leq 1/(2^N - (i-1))$. $P((I,C) \leftarrow A^\kappa(\sigma) \mid \kappa(I,C) = \sigma) \leq P(S_1 \vee S_2 \vee \cdots \vee S_q) \leq \Sigma_{i=1}^q P(S_i) \leq \Sigma_{i=1}^q \frac{1}{2^N-(i-1)} \leq \frac{q}{2^N-2^{N-1}}$ for $q \leq 2^{N-1}$. This is bounded by $q/2^{N-1}$. □

# 4 Application

The primary application for the proposed compression functions is memory integrity in tamper-evident architectures. A tamper-evident architecture provides an execution environment for a program which detects any tampering. AEGIS [8] offers such an architecture. XOM [4] is an example of tamper-resistant architecture, which lacks in memory integrity verification. Memory integrity verification [3] is provided with Merkle hash trees. We first define memory integrity property. A processor communicates with memory $M$. Memory $M$ has two attributes, addresses $A$ and contents $V$. It maintains associations between addresses and contents. A read of memory at address $A$ denoted by $M[R,A]$ returns the value associated with $A$. A write into memory address $A$ of value $V$ is denoted by $M[W,A,V]$. A write of $A$ with value $V$ immediately followed by a read of address $A$ must return value $V$. If we need to argue about temporal sequences of reads and writes, we will need to associate time $T$ with reads and writes as $M[R,A,T]$ and $M[W,A,V,T]$.

**Definition 4.1** (memory integrity)**.** *A read of address $A$ at time $T$ should return the value written to address $A$ at time $T' < T$ such that no other write to $A$ occurs between time $T'$ and $T$. In other words: $M[R,A,T] = V$ iff $\exists M[W,A,V,T']$ for $T' < T$ and $\forall t \in [T'+1, T-1]$, $\nexists M[W,A,*,t]$. We can assume that each address has a write to it at time $T = 0$ of value 0.*

A Merkle hash tree of address range $[A,A+k]$ creates a tree of hashes with $k+1$ leaves corresponding to addresses $A, A+1, \ldots, A+k$. Any write to an address $A+i$ in this address space can modify all the hash values from the leaf node corresponding to $A+i$ upto the root of the tree. Any read from address $A+i$ needs to check the hash values along the path from the leaf node $A+i$ upto the root of the tree. The root of the tree is stored in the trusted processor storage, so that it cannot be tampered. All the other tree nodes along with the leaf nodes can be kept in the untrusted memory. The overhead of such integrity verification architecture is several hashing operations, $\log N$ for $N$ leaf nodes. One can cache some of these hash tree nodes to increase the efficiency. The granularity of a leaf node can be increased beyond a single word to an entire cache block. Despite these optimizations, such hash trees are expensive primarily due to the cost of the underlying hash function cost. AEGIS [8] charges 160 cycles for each hashing operation presumably at a cost of 2 cycles per round for 80 rounds of SHA. This is a very high cost for a memory integrity architecture that

spawns many hash function instantiations for each read and write. AEGIS optimizes by assuming that a large Level-2 cache is part of the trusted processor boundary. This allows one to perform memory integrity verifications only when a read or write crosses the L2 cache to main memory boundary. Even with that, the performance penalty is of the order of 25%. In an embedded processor, L2 cache in not even likely to be integrated into the trusted processor core. Hence, more efficient mechanisms for hashing memory contents are desirable.

**Adversary Model:** The traditional adversary model for a hash function collision resistance, preimage and second preimage resistance assumes that the adversary has oracle access to the hash function. This is a valid model for a publicly available hash function $H_K$ instantiated with a secret key $K$, which can be repeatedly exercised by an adversary. In a trusted processor, where a personalized hash function per process can be selected, the adversary does not have a mechanism of exercising this hash function, unless s/he has control of the executing process. In such a case, the adversary has no need to tamper with the memory resident data! An outsider adversary, one who does not control the process, can only observe hashed values, $H_K(V)$, of the Merkle tree stored in the untrusted memory. Hence, the relevant problem for the adversary is to guess the real memory contents associated with the hash. The adversary may also have access to the memory contents whose hash is under attack. In memory integrity verification architectures, however, the memory contents are also encrypted with a block cipher function such as AES. Figure 3 shows this adversary model. Given this, we assume that the adversary's goal is to derive the preimage of a hash value.

The proposed family of compression functions $\mathcal{K}(64,4)$ offers good preimage resistance as shown in Theorem 3.1 which is all that is needed for the memory integrity verification.

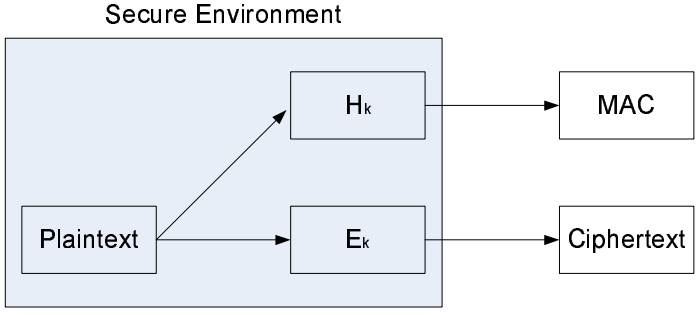


Figure 3: Memory Integrity Verification Adversary Model

### 4.1 Encryption

One can construct a block cipher function $\kappa(I,C)$ from $\mathcal{K}(64,4)$ as a Feistel network [2]. In these cases, the input $C$ can serve as a key to select a specific function $\kappa_C : \{0,1\}^N \rightarrow \{0,1\}^N$. Two stages of Feistel network may suffice for such a construction. We need to analyze the properties of such an encryption function.

### 4.2 Hash Function

If one views $\kappa_C : \{0,1\}^N \rightarrow \{0,1\}^N$ as a compression function, then iterated hashes of Preneel, Govartes, Vendewalle [6] from any of the Group 1 schemes [1]. Collision resistance and second preimage resistance of $\kappa_C$ needs to be analyzed for this application.

## 5 Conclusions

We started out with the goal of developing a inversion resistant function with a low latency and low area hardware implementation. We based such a function on two properties of truth tables: balance and independence. We demonstrated that a composition schema of unbiased and independent 4-input gates leads to unbiased and independent compression functions.

## References

[1] John Black, Phillip Rogaway, and Thomas Shrimpton. Black-box analysis of the block-cipher-based hash-function constructions from pgv. In *CRYPTO '02: Proceedings of the 22nd Annual International Cryptology Conference on Advances in Cryptology*, pages 320–335, London, UK, 2002. Springer-Verlag.

[2] H. Feistel. Cryptography and computer privacy. *Scientific American*, 228(5):15–23, May 1973.

[3] B. Gassend, G. Suh, D. Clarke, M. van Dijk, and S. Devadas. Caches and merkle trees for efficient memory integrity verification. In *Proceedings of Ninth International Symposium on High Performance Computer Architecture*, 2003.

[4] David Lie, Chandramohan A. Thekkath, Mark Mitchell, Patrick Lincoln, Dan Boneh, John C. Mitchell, and Mark Horowitz. Architectural support for copy and tamper resistant software. In *Architectural Support for Programming Languages and Operating Systems*, pages 168–177, 2000.

[5] R. C. Merkle. Protocols for public key cryptography. In *Proceedings of the IEEE Symposium on Security and Privacy*, 1980.

[6] Bart Preneel, Ren Govaerts, and Joos Vandewalle. Hash functions based on block ciphers: A synthetic approach. In *Proceedings of Crypto '93*, volume Lecture Notes in Computer Science, Volume 773, pages 368–378, 1994.

[7] Phillip Rogaway and Thomas Shrimpton. Cryptographic hash-function basics: Definitions, implications, and separations for preimage resistance, second-preimage resistance, and collision resistance. In *Proceedings of Fast Software Encryption: 11th International Workshop, FSE 2004*, volume LNCS-3017, pages 371–388. Springer-Verlag, 2004.

[8] G. Suh, D. Clarke, B. Gassend, M. van Dijk, and S. Devadas. aegis: Architecture for tamper-evident and tamper-resistant processing. In *Proceedings of the 17 Int'l Conference on Supercomputing*, pages 160–171, 2003.