# Blackbox Adversarial Learning on Hash Functions

Anirban Gangopadhyay
*Your Institution*

Joshua Zweig
*Second Institution*

## Abstract

Your abstract text goes here. Just a few facts. Whet our appetites. Not more than 200 words, if possible, and preferably closer to 150.

We present a novel machine learning framework which we apply here to uncover weaknesses in hash functions. Our framework extends the definition of adversarial learning to the domain of cryptographic systems. In this paper, we uncover weaknesses in Merkle?Damgard structured hash functions, which include MD5, SHA-1 and SHA-2 and are critical components of cryptographic systems such as key exchange, digital signatures, and password and file verification. Most attacks against these hash functions leverage properties of the particular function. We develop a broader framework of adversarial learning to probabilistically determine how a hash function differs from its ideal state. We show the efficacy of our framework by theoretically and empirically discovering known vulnerabilities in MD5 and discuss how our method can be applied to uncovering potential unknown weaknesses in other hash functions in the Merkle?Damgard class.

## 1 Introduction

Talk about the purpose of the paper. Emphasize how our work is novel.

**Note:** Traditional adversarial learning is defined as a technique employed in the field of machine learning which attempts to fool models through malicious input. Our definition is slightly different so we need to be deliberate in explaining how our new definition still ties to this concept

### 1.1 Related Work

May or may not want this as a subsection. Including for now.

## 2 Contributions

- Expand definition of adversarial learning to a broader class of problems
- Leverage this definition of adversarial learning to introduce a black box attack on hash functions of the Merkle construction
- Uncover vulnerabilities in MD5 with this methodology (TBD based on results how we'll frame this)
- Epsilon, Delta framing

## 3 Background

Do we need this section in order for the paper to be self contained?

## 4 Adversarial Learning on Cryptographic Systems

In this section we extend the definition of adversarial learning and explore its relevance in the context of cryptographic systems. We note that all cryptographic systems rely on the notion of randomness and intractability. Hence, by framing an adversarial set of examples as one that yields a reduction in entropy on the outputs, we set ourselves up to empirically discover inputs that violate the properties of cryptographic functions.

### 4.1 Adversarial Learning

We start by presenting the notion of adversarial learning in the context of cryptographic systems. In our construction, we treat the system as a blackbox $B(\cdot)$ and consider the following problem of finding adversarial examples:

**Meta Problem:** Given a blackbox system $B(\cdot)$ that takes in as input $X : \{0,1\}^M \to Y$ (where $Y \in R^N$), can we find a subset $X_{adv}$ such that the error function $F_\varepsilon(Y, Y_{adv}\}$ is large for some error function $F_\varepsilon(\cdot)$ that measures a difference in entropy over the output sets.

We note that a reduction in entropy on the output set can be measured in different ways. Salient examples include a reduction in Shannon entropy on $\hat{Y}$, higher predictability of the distribution formed by $\hat{Y}$, or a parametrization of $\hat{Y}$ via a set of discriminating features.

The type of entropy reduction we use depends on the construction of our blackbox system $B(\cdot)$, and the specific properties of randomness we would like to exploit.

We give specific definitions of $F_\varepsilon(\cdot)$ for the examples discussed above.

- Reduction in Shannon Entropy: We define $F_\varepsilon(\cdot) = \frac{E[H(Y_{adv})]}{E[H(Y)]}$ where $H(Y) = -\sum_{i=1}^{n} P(y_i) log P(y_i)$.
- Predictability of Distribution: We define $F_\varepsilon(\cdot) = KL(Dist(\cdot)|P(Y_{adv}))$ where $KL(\cdot)$ represents KL divergence - a measure of distance between probability distributions, $Dist(\cdot)$ represents the distribution representing $P(Y_{adv})$. We note $P(Y_{adv})$ represents the probability distribution over the adversarial set. We also note $Dist(\cdot)$ must be nonuniform for our examples to be truly adversarial.
- Parametrization - We define $F_\varepsilon(\cdot) = F_\Theta(X_{adv})$ where $\Theta$ represents a parametrization over the inputs, yielding the output set $Y_{adv}$.

Each definition of $F_\varepsilon(\cdot)$ lends itself to various attack vectors that allow for the exploitation of $B(\cdot)$. Hence, we consider the class of adversarial examples to be inputs that enable entropy reductions such as the ones specified above.

## 4.2 Cryptographic Systems

What does this look like in the context of PRNGs, public key generators, hash functions?

## 5 Adversarial Learning on Hash Functions

**Meta Problem for Hash Functions:** Can we quantify how much a hash function deviates from its ideal state (perfect one way function)?

### 5.1 Notion of Blackbox

Define w.r.t to threat model. This isn't zero knowledge, an attacker needs to know the construction is Merkle construction

-

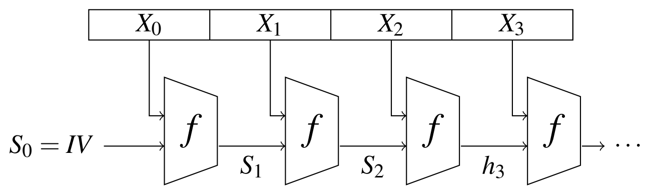
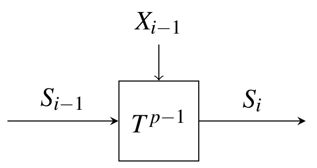


Figure 1: The Merkle–Damgård construction of secure hash functions.

### 5.2 Blackbox Definition

Define the blackbox. Explain how it is general enough due to Merkle process to apply to MD5, SHA1, SHA2



For the Merkle–Damgård class of hash functions, we define a singular blackbox component, $B$ as the function $S_i = B(S_{i-1}, x_{i-1})$. We will outline was this definition is sufficient to model any Merkle–Damgård function later in this section.

We limit the scope of the empirical portion of our study to the Merkle–Damgård class of hash functions, which includes MD5, SHA-1, and SHA-2. We limit the scope in this way to be able to give a definition for a blackbox component $B$ towards validating our framework, noting that our methods remain general and applicable to other contexts and for different definitions of $B$.

The Merkle–Damgård construction is detailed in Figure **??**. In this construction, some padding is applied on the input message to generate $X = X_0 X_1 X_2 \cdots X_n$, where $n$ is the number of chunks the input is broken into. A function, $f$, is applied to each of these chunks, $X_i$ and also to the state information output from the previous step, $S_i$. The state is seeded in some way for any particular function conforming to the Merkle–Damgård construction. The primary creativity any particular hash function can apply is in the definition of the seed, the padding function and the definition of $f$. Each particular hash function, though, must define some function $f$ which takes as an input a chunk of the input message $X_i$ and state information from the previous state and only the previous state, $S_i$, and output its own state information for consumption by the next application of $f$. We model the blackbox component $B(S_{i-1}, x_{i-1})$ to assume these properties and only these properties. The model remains agnostic to any particular padding function, seed, and especially definition of $f$ as defined by any conforming hash function, be it MD5, SHA-1, or SHA-2. This implies that this methodology, unchanged, can be freely applied to any of these three hash functions towards analyzing their weaknesses.

## 5.3 Markov Process

We discuss our machine learning framework which is designed to detect weaknesses in our blackbox function $B(\cdot)$ for Merkle–Damgård structured hash functions. Our goal is to determine if a hash function is weak given an example set $X'$. We note that the states outlined in our framework are purely Markov states and dont represent any transform in the hash function itself. We specifically favor this blackbox construction since it generalizes to any Merkle Damgard hash construction.

We denote the set of Markov states to be the complete set $(S_i)$ such that $S_i \in \{0,1\}^n$ where $n$ varies by hash function. We parametrize this set within the context of blackbox components $B = \{B_1, ..., B_N\}$ where $N = 2^n$.

A transition from $B_i$ to $B_j$ is represented as an input chunk-state pair $(S_{i-1}, X_i)$ yielding the output state $S_i$.

Given a set of inputs $X'$, we train our Markov states by considering the sequence of chunk transforms $S_0 \to B^1(X_i, S_i) \to S_1, ..., \to S_{p-1} \to B^P(X_i, S_i) \to S_p$ where $P$ is the number of steps in a round of $B(\cdot)$ for each $X$.

By using MLE estimates and applying the Markov assumption, we get $P(B^i|B^j) = \frac{count(B^i(X_i,S_i) \to S_j)}{count(B^i(X_i,S_i)}$. **Note:** Can provide proof if necessary

Hence, we can represent our Markov transition probabilities via a $N \times N'$ probability matrix where $N' = 2^n$, $N$ is defined as above.

Our goal is to identify given a set $\hat{X}$, whether $\frac{E[P(\hat{B})]}{E[P(B)]} < \varepsilon$ for some predefined $\varepsilon$. If so we denote $\hat{X}$ adversarial.

We can define $P(B)$ as the Shannon entropy over the elements of the state transition matrix $S$. Specifically, we find the distribution of element values and then compute $H(S) = -\sum_{i=1}^{n} P(s_i) log P(s_i)$.

Define the Markov process, how this can be used to identify weak hash functions. Tie in the generative discussion here

The current notion of a Markov state is defined as an input of a blackbox component ($\{0,1\}^{128}$). Can we define a state more coarsely (parametrized in some way). Ideas:

- Collapse Markov state in a cascading manner (Random forest analogy)

## 5.4 Finding Adversarial Examples

Define the sampling scheme for finding adversarial examples.

First define Theorem 1 (state as the naive sampling method).

Then state our novel sampling method along with Theorem 2

## 5.5 Finding Weak Hash Functions

Here we outline our specific algorithms for finding:

- Collisions
- Correlations
- Preimage

The presentation of the algorithm should follow the following structure:

1. Provide a mathematical definition of the weakness
2. Outline the algorithm (using the one from previous section) for uncovering the weak property
3. Expected number of examples to uncover the weak property in a vulnerable hash function
    - Consider the generative process or statistical properties that a given weakness enables

## 5.6 Validating Adversarial Examples

Outline how domain based methods have uncovered specific weaknesses in MD5 (and if time SHA1), how to characterize the input and output sets of the weakness (show that the statistical properties that a given weakness described in the previous section apply), and how our algorithm ties to the same set of inputs and outputs (but using a completely different methodology)

Also describe how our algorithm is novel and can be applied to any hash function to test weakness. We simply validate this with MD5 (and perhaps SHA1)

## 6 Results & Applications

1. Seed sampling algorithm with known collisions and see if other collisions are uncovered. Logic is based on the fact that tunneling properties provide necessary but not sufficient conditions for collisions
2. Can test with SHA1 as well
3. Run the sampling scheme for different values of $\varepsilon, \delta$ and see the collision rates, yields of $\hat{X}$

4. Derive $m$ for Theorem 1 given different values of $\varepsilon, \delta$ and map that to reasonable CPU measures. Then test this empirically and validate our values of $\varepsilon, \delta$ measure up empirically (Frame in terms of threat model)

5. Run on known good hash functions (ie SHA256) and see what results we get

**Figure out:** What charts do we want to show? How do we want to structure the results section in the most effective way possible? Options are:

1. Table of $m, \varepsilon, \delta$, CPU measures for naive algorithm, Theorem 2 alg, alg for uncovering each weakness

2. A chart of CPU/cores vs time it took to run

3. Can have bar charts that illustrate how many times (over N runs), we found a collision, etc

4. Replicate these for the case where we seed the sampling algorithm with known collisions

## Related Work

Maybe we can work in sufficiently elsewhere

## Acknowledgments

The USENIX latex style is old and very tired, which is why there's no \acks command for you to use when acknowledging. Sorry.

## Availability

USENIX program committees give extra points to submissions that are backed by artifacts that are publicly available. If you made your code or data available, it's worth mentioning this fact in a dedicated section.