Adversarial Learning on Blackbox Hash Functions

Anonymous Anonymous

Abstract

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–Damgård structured hash functions, which includes MD5, SHA-1 and SHA-2. which are critical components of modern cryptographic systems such as those of key exchange, digital signatures, and password and file verification. Most attacks against these hash functions leverage properties of the particular function, while the framework we present applied to the construction itself. We develop a 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 weaknesses in other hash functions in the Merkle–Damgård class, as well as other types of cryptographic systems.

1 Introduction

There have been many domain based efforts that have attempted to find vulnerabilities in hash functions and cryptographic systems [2, 6, 7, 11, 16, 30, 34, 36]. These attacks are in general highly dependent on specific properties of the system or protocol. With respect to hash functions, Wang seminally uncovered vulnerabilities in MD5 and there has been much follow on work in applying novel methods to uncover weaknesses in hash functions, public key generators and the like [10, 35, 37]. Much of the work in this domain has either been of a cryptographic nature [5, 33], exploiting mathematical weaknesses in the algorithms - or of an information security nature, exploiting weaknesses in the protocol design and implementation. Although these methods have the benefit of solid mathematical grounding and ease of reproducibility, due to the domain specific nature of such work, it has been very difficult to generalize such approaches to structurally finding weaknesses in cryptographic systems.

We draw inspiration from the field of adversarial learning in machine learning and present a framework that can be used to assess the weaknesses of cryptographic systems and potentially uncover new vulnerabilities. Much work has been done in the study of finding adversarial examples against machine learning systems, including for blackbox machine learning systems. Goodfellow introduced seminal work in this domain and defined an adversarial example to be malicious input x that fools a model into outputting a misclassified label y' (where the true label was y) [8, 19].

We expand the definition of adversarial learning to apply to the cryptographic context. This includes rigorously defining what it means for a given set of examples to be *adversarial*, and justifying how the adversarial set can lead to uncovering weaknesses. We then apply this framework to the task of uncovering weaknesses in hash functions. We do this by constructing the notion of a blackbox for Merkle–Damgård structured hash functions and present a framework that finds and validates adversarial examples. Lastly, we tie our framework back to the notion of weak properties as they apply to a hash function. Namely, we show how our results lead to the uncovering of collisions. We also describe how this framing and result could apply to correlative and preimage attacks in this context.

Our framework has the benefit of modularizing the process of defining, uncovering and validating adversariality. Hence, although we present an end to end pipeline specifically for Merkle–Damgård structured hash functions, our framework can be applied to any cryptographic system that fits within our blackbox paradigm. We specifically present results in the context of the MD5 algorithm due to ease of comparison but we discuss how our methods can be applied similarly to the class of Merkle–Damgård structured hash functions including the SHA1 and SHA2 algorithms.

1.1 Related Work

We outline related work that has been conducted in the areas of exploiting hash function weaknesses, and using machine learning against cryptographic systems. [14, 24] conducted work in analyzing the hardness of hash functions. This paper introduced the notion of hash functions with regards to satisfiable and unsatisfiable problems within the NP hard class, establishing a framework to evaluate the class of hash functions. There is a class of work [4, 5, 7, 20, 28–30] that focuses on domain based attack mechanisms against hash functions. This work focuses on precise and deterministic attacks against specific hash functions and aims to bring down the intractable nature of the problem. Although these attacks are easily reproducible and clearly uncover vulnerabilities, they are not generalizable and seldom can be used to infer weaknesses outside of what was uncovered.

Much work has been done in relating machine learning methods to cryptography and cryptographic systems. Rivest's seminal paper on the topic [24] outlines the mathematical and semantic relationships between the two fields. There have been many studies that attempt to use machine learning to find mathematical or implementation vulnerabilities in cryptographic systems [9, 12].

Due to the intractable nature of primality, there has not been much successful work in applying machine learning methods to uncover vulnerabilities in cryptographic systems. A further drawback of machine learning methods is the lack of reproducibility due to the empirical nature of the field, thus requiring replication of data and parameters to uncover interesting results. The framework we posit in here gives a structure within which future research in the field can innovate. Additionally, the hashing problem we explore is particularly well suited for machine learning based approaches. We further note that this is particularly the case with reference to our frameworks ability to address the reproducibility and explainability problems.

2 Contributions

In this section we outline our contributions.

- **Expanded definition of adversarial learning to a broader class of problems** We consider the notion of adversarial learning against machine learning systems and expand this to cryptographic systems. The novelty of our expansion is multifold. Not only do we apply this concept to a new domain, but we robustly define a notion of adversarial in an unsupervised context (without (x, y) labels)
- **Leverage our definition of adversarial learning to introduce a black box attack on hash functions of the Merkle–Damgård construction** We define a blackbox construction for the class of Merkle–Damgård structured hash functions that abstracts differences within MD functions but codifies features that apply to the class. This allows us to construct a machine learning approach that explores properties across the Merkle–Damgård class.
- **Construct a Machine Learning Approach to Finding Adversarial Examples against Merkle Hash Functions** We present a novel machine learning framework that optimizes towards finding adversarial examples. We then tie the application of our framework to uncovering collisions. We note this contribution, developing a machine learning approach to uncovering weaknesses, as distinct from the previous contribution of developing a blackbox framework from which to reason about an attack.
- **Theoretical Bounds on Performance** We introduce theoretical bounds on how many examples we expect to generate before finding an adversarial set. These bounds are applicable to applications of our framework beyond hash functions.
- **Uncover collisions in MD5** We successfully apply our framework to discover collisions in MD5.

3 Background

In order for this paper to be self contained, we include this section with background information required by our methodology. It additionally serves as a compliment to the previous section on related work.

3.1 Background on Hash Function Construction and Weaknesses

There are three elements of hash functions that are important to our approach. These include (1) the properties of cryptographic hash functions, (2) the Merkle–Damgård construction of secure hash functions, and (3) known structural attacks against hash functions of this construction.

Cryptographic hash functions are the "workhorses of modern cryptography" [26]. These hash functions take an arbitrary length string as input, and deterministically produce a fixed length output. It is the goal of these hash functions to make it very hard to find to inputs X, X' such that $H(X) = H(X')$. However, since the output is fixed length, but the input length is arbitrary, it is necessary that some collisions must exist. For example, if the output of $H(X)$ for some hash function, H is b bits, and an attacker computes $H(X)$ for $2^b + 1$ values of X, the attacker will find a collision. Finding two values X, X' such that $H(X) = H(X')$ is known as a second preimage attack or a collision attack.

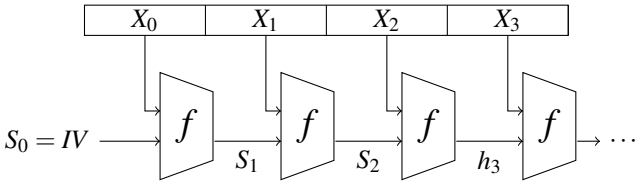


Figure 1: The Merkle–Damgård construction of secure hash functions.

Cryptographic hash functions should also be resistant to preimage attacks, given a hash value h finding an input X such that $H(X) = h)$ should be difficult. Additionally, these functions should be resistant to correlation attacks, that is a set of correlated inputs X should have hash values that appear random and uncorrelated.

The most popularly used cryptographic hash functions today all use the same construction, the Merkle–Damgård construction [3,21,26]. The Merkle–Damgård construction is detailed in Figure 1. In this construction, some padding is applied on the input message to generate $X = X_0X_1X_2\cdots X_n$, 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 (we will introduce the index j in 5.2.1). The state is seeded in some way for any particular function conforming to the Merkle–Damgård construction.

In addition to the states S in Figure 1, there are generally additional state transitions within any function application block f, as is the case for MD5, SHA1 and SHA2.This will be further explored in section and a notion important to our methodology.

Lastly we describe a structural attack that has been applied to multiple Merkle–Damgård hash functions. Wang, et. al. [35]. This attack works by changing any two consecutive input blocks such that the state outputs from the first blocks force a near collision and that the state outputs from the second blocks force a collision. This leverages the fact that in the Merkle–Damgård construction, only the state information is carried to the next step. This structure has been used successfully in finding many collisions and will be again levered in applying of our methodology to MD5.

3.2 Background on Machine Learning and Adversarial Attacks

Adversarial learning is a relatively new area of machine learning, but has quickly garnered popularity, for example with the widespread use and impressive results that have been produced by GANs. Historically, adversarial learning has been applied to machine learning classifiers and aims to find adversarial examples that trick the classifier into mislabeling the output. Specifically, adversarial examples x_i^{adv} are slightly perturbed examples from x_i such that the label y_i^{adv} is significantly different than the label y_i.

Adversarial examples pose significant security threats within the context of machine learning models due to the property of transferability. Specifically, an adversarial example that is misclassified by a given model is often also misclassified by a similar model. This opens up attack vectors against machine learning blackbox systems [23]. As a result there has been a recent flurry of study around the notion of adversarial learning and how it applies to real world machine learning systems. The focal points of such studies orient around building model robustness against such examples, understanding different whitebox and blackbox attack vectors, and understanding how adversarial examples can disrupt the flow of real world systems. Studies such as **??** have conducted experiments to simulate real world examples of adversarial vulnerabilities as they exist in the real world.

Although the advent of adversarial learning has been an exciting one, the field has so far been limited to focusing purely on machine learning systems. Furthermore, the notion of an adversarial example very specifically alludes to an input x_i^{adv} that is slightly perturbed from a benign input and yet produces a significantly different label. To that end much of the research [1, 31, 32] has been localized to studying examples of this property and building robustness against adversarially perturbed inputs. This includes injecting adversarial examples into training data and intelligently training models to specifically learn properties of such examples [13].

In this paper we aim to extend the definition of adversarial learning to uncovering sets of examples that expose vulnerabilities in cryptographic systems. To this end, we define weakness as it applies generally to cryptographic systems, ground our definition in multiple examples of systems. Finally we outline a process for discovering and evaluating adversarial examples as they apply to hash functions.

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 $X : \{0,1\}^M \rightarrow Y$ (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 $F_\varepsilon(\cdot) = \frac{H(Y_{adv})}{E[H(Y)]}$ 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 Θ 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

We discuss our notion of adversarial learning in the context of general cryptographic systems. Specifically we consider pseudo random number generators, public key generators and hash functions. We present our adversarial framework for PRNGs and public key generators. The remaining parts of the paper outlines in detail the notion of adversarial learning specifically within the context of hash functions.

PRNGs: We first consider the class of pseudorandom number generators and define the notion of an adversarial set in this context. We note a given PRNG $F(\cdot)$ is a function that takes in a seed s and outputs a sequence of numbers whose properties approximate the properties of sequences of random numbers.

Assuming we define our blackbox to be $F(\cdot)$, our definition of an adversarial examples very naturally maps to the class of PRNGs. Specifically, if we assume $E[H(Y)]$ to be the expected entropy over a randomly generated sequence of numbers Y of size M, we can define an adversarial seed s to be one that produces a sequence of numbers Y_{adv} such that $H(Y_{adv}) < K \cdot H(Y)$.

Public Key Generators: We consider the blackbox system $B(\cdot)$ to be the generation of the public key and private key pair. This is used for public key encryption algorithms including RSA [15]. The algorithm works as follows:

1. Choose two distinct prime numbers p and q
2. Compute $n = p \cdot q$. The length of n (in bits) is going to be the key length.
3. Compute $\phi = (p-1) \cdot (q-1)$.
4. Choose the exponent e (usually 65537)
5. Compute $d = e^{-1} \mod \phi$
6. Public key is the pair (n, e)
7. Private key is the pair (n, d)

Note in this blackbox system, we do not feed in an input X_i to $B(\cdot)$ and get Y_i but rather $B(\cdot)$ spits out (X_i, Y_i) pairs.

We define X, Y, entropy $H(Y)$ for this blackbox system $B(\cdot)$.

- X is defined to be the public key pair (n, e) denoted p_b
- Y is defined to be the private key pair (n, d) denoted p_k.
- The **vulnerability** can be defined as recovering p_k given p_b
- $H(\hat{Y})$ for the set $\hat{Y}$ is defined to be some function of how easy it is to GCD pairs of public keys and retrieve private keys (ie measure the entropy of GCD pairs)

contrast to domain based methods

5 Adversarial Learning on Hash Functions

We present our framework for adversarial learning against hash function. Our approach pivots around the following meta problem:

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

We construct a blackbox model that represents the Merkle–Damgård class of hash functions and then outline our machine learning framework which intelligently generates examples and measures how these examples lead to outputs that deviate from entropy values of the ideal state.

5.1 Notion of Blackbox

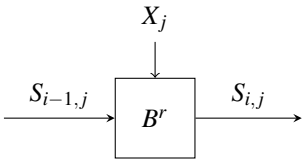
We present a blackbox attack on the Merkle–Damgård class of hash functions. We write *blackbox* to mean that the attack does not depend on the parameters or implementation details of a particular hash function. This allows our methodology to be trivially applied to any hash function of the Merkle–Damgård construction.

Most research on finding weaknesses in hash functions focuses on finding weaknesses in a particular hash function. For instance, take the years of research focused on finding collisions in MD5 [17, 18, 20, 35] and the recent attack which successfully found the first full collision for SHA-1 [27]. In general, this research focuses on determining a set of necessary conditions that any inputs resulting in a collision must meet. These sufficient conditions reduce the state space of hashs that need to be computed to find a collision. Once the state space is small enough that it can be searched, a collision can be computed [25]. Deriving the sufficient conditions depends on observations that are made about the parameters of a particular function. This prevents such an attack from being applied to any other hash function.

The paradigm we present here does not depend on the specific implementation of any hash function, only that it is of the Merkle–Damgård construction. For this reason, the attack can be applied with ease to MD5, SHA1 or SHA2. By *blackbox* we do not mean that an attacker does not know the particular hash function they are attempting to find weaknesses in, though they may not have this information. We specifically mean that that the attack does not depend on the implementation details of the hash function and that an attacker therefore does not need to know them to leverage this attack.

5.2 Blackbox Definition

For the Merkle–Damgård class of hash functions, we define a singular blackbox component, B, as the function $S_{i,j} = B^r(S_{i-1,j}, x_j)$. We will outline how 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, SHA1, and SHA2. 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 primary creativity any particular hash function of the Merkle–Damgård construction 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. In many hash functions of this construction, including MD5, SHA1, and SHA2, f is a piecewise function of the step of the hash function.

We model the blackbox component $B(S_{i-1,j}, x_{i-1})$ 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, SHA1, or SHA2. This implies that this methodology, unchanged, can be freely applied to any of these three hash functions towards analyzing their weaknesses.

5.2.1 Reduction to MD5

In this section we map our blackbox construction to the particular implementation of MD5. MD5 has 4 rounds and 64 steps. Thus, where $p \in [0, 63]$ is the step in a single MD5 block f can be defined as

$$f(p) = \begin{cases} F & p \leq 15 \\ G & 15 \leq p < 32 \\ H & 32 \leq p < 48 \\ I & p \geq 48 \end{cases}$$

Having given f, it is left only to define B. We define 4 blackbox components, B^r where $r \in 1, 2, 3, 4$, one for each round

in MD5. For example, B^2 captures transformation where $p \in [13, 32)$ We further define $S_{i,j}$ where i is the number of chunks x_i in the input X as defined in 5.2, and $j \in [0, p]$. Thus, we capture the state at each of the p steps for a single i. Thus, for example, the output state for the ith block is denoted $S_{i,p}$.

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 of inputs $\hat{X}$.

We posit a distribution over the states z_{i+1j} as a function of z_{ij} and x_j. We note the distribution represents a generative process that moels B. We make the following conditional independence assumptions about $p(z_{i+1j})$:

$$p(z_{i+1j}) = p(z_{i+1j}|z_{ij}) \cdot p(z_{i+1j}|x_j).$$

We note $p(z_{i+1j}) = p(z_{i+1j}|z_{ij}, z_{i-1j}, ..., z_{0j})$. Given the Markov assumption, $p(z_{i+1j}|z_{ij})$ is only conditionally dependent on z_{ij}. Taking the maximum likelihood, we derive $p(z_{i+1j}|z_{ij})$ as:

$$p(z_{i+1j}|z_{ij}) = \frac{count(z_{i+1j}, z_{ij})}{count(z_{ij})}.$$

We can similarly derive $p(z_{i+1j}|x_j) = \frac{p(z_{i+1j} \cap x_j)}{p(x_j)}$

which implies $p(z_{i+1j}|x_j) = \frac{count(z_{i+1j}, x_j)}{count(x_j)}$ by MLE derivations.

Therefore, we obtain our discrete probability distribution function that posits over our states z_{ij} and input chunks x_j. We next define our state space and enumerate our training process.

5.3.1 State Space

We start by defining our state space Z. Since our random walk is defined over the transition of steps, $S_i = B(S_{i-1}, x_{i-1})$, we consider the span of Z to be over all possible instances of S_i. Since $S_i \in \{0,1\}^M$ for some fixed M (note M varies by hash function), the cardinality of Z is 2^M. Similarly the state space of x_{ij} is over $\{0,1\}^N$ where N denotes the size of the input chunk for our given hash function.

5.3.2 Training our PDF

Given a set of inputs $\hat{X}$, we train our probability distribution function by considering the sequence of step 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 $P(z^i|z^j) = \frac{count(B^i(x, S_i=z_i) \to S_j=z_j)}{count(B^i(x, S_i=z_i)}$.

Similarly, we obtain $P(z^i|x^i) = \frac{count(B^i(x_i, S_i=z_i))}{count(B^i(x_i, S)}$

Hence, we can represent our Markov transition probabilities in matrix form and derive $p(z_{ij})$ accordingly. We note that the size of our transition matrix for $P(z^i|z^j)$ and $P(z^i|x^i)$ are $M \times M$ and $M \times N$ respectively which are intractable for most hash functions. We note that for the purposes of finding weaknesses in our hash function, we dont need to learn the entire distribution but rather only a locally adversarial subset. We apply Laplace smoothing to initialize each element of our transition matrices via constant $\frac{\eta}{N^2}$ and $\frac{\eta}{N \cdot M}$ respectively. These represent uniform priors on the unobserved transition states. As we train our transition matrices, we renormalize the matrices to preserve stochasticity.

5.3.3 Adversarial Set

Our goal is to identify given a set $\hat{X}$, where $\frac{P(\hat{Z})}{P_{ideal}(Z)} < \varepsilon$ for some predefined ε. If so we denote $\hat{X}$ adversarial. We note that $P_{ideal}(Z)$ denotes the transition matrix if the hash function were in its ideal state. We derive $P_{ideal}(Z)$ below based on the assumption of a perfect one way function.

We start by providing the definition of a perfect one way function.

One Way Function: A function $f : \{0,1\}^* \to \{0,1\}^*$ is strongly one way if the following two conditions hold:

1. There exists a polynomial-time algorithm, A, such that given input x, A outputs f(x) (i.e f(x)=A(x))

2. For every probabilistic polynomial-time algorithm A', every polynomial p() , and all sufficently large n's, $P[A'(f(x)) \in f^{-1}f(x)] < \frac{1}{p(n)}$.

From a generative perspective, we can also view a perfect one way function as returning an output $f(x)$ uniformly over the set of outputs, given any input x. We note that although a one way function doesnt require $f(x)$ to be generated from a uniform distribution, it requires $f(x)$ to be non-distinguishable from a uniform distribution. Hence, we derive $P_{ideal}(Z)$ as the entropy over the set of states M such that $p(z) = \frac{1}{M}$ for all $z \in Z$.

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

5.3.4 Generative Assumption

Our Markov model can be viewed as a generative model that parametrizes the step transitions of the Merkle–Damgård construction. In its ideal state, a hash function would have uniform probability across all transitions. So $(P(z_i|z_j)$ would be uniformly probable for all i, j in $|Z|$. The further the transition matrix deviates from uniform, the lower the entropy within the state of inputs. Hence, we can view our matrix as a parameter over the Multinomial distribution that measures entropy of the state space given a set of inputs $\hat{X}$.

5.4 Finding Adversarial Examples

In this section, we consider the problem of determining if a given hash function represented as $B(\cdot)$ is weak. We first present a theorem that establishes bounds on the expected number of examples m that need to be drawn before we can determine if $P(Z)$ deviates significantly from the ideal state $P_{ideal}(Z)$. We then present our algorithm for uncovering an adversarial set X_{adv}, where X_{adv} is a subset of X and $|X_{adv}| << |X|$.

5.4.1 Weakness Detection

Consider the following theorem that, given the ability to query a subset $\hat{X}$ from X, identifies the number of samples m needed (with probability δ) to determine if $P(Z)$ deviates enough from its ideal state.

Theorem 1 *For a given ε, δ the expected number of examples m needed such that $P(\frac{H(Z)}{H_{ideal}(Z)}) < \varepsilon$ is $m = F(\varepsilon)$ for the state space Z where the set of states are defined in our Markov model.*

We derive $m = F(\varepsilon)$ and present the proof below. Our proof follows an inductive structure. We note that $H(Z) = -\sum_k p(z_k) log p(z_k)$. We also note after a certain number of examples m_0, there is a high probability that the entropy goes down as we observe a new example. We derive how much the entropy is expected to go down with each example.

We wish to derive $E[H(Z_{new}) - H(Z_{old})]$. Let us denote the size of $Z_{old} = m$ and $|Z| = M$.

This is equivalent to $E[-\sum_{k*} p(z_k^*) log p(z_k^*) + \sum_k p(z_k) log p(z_k)]$.

We note the presented paradigm represents the naive approach of sampling the hash function m times and measuring the change in entropy of the states.

We note that if a new state is added to the set then the entropy goes up (this happens with $p = \frac{(M-m)}{m}$). Otherwise the entropy goes down.

Specifically, $E[H(Z_{new}) - H(Z_{old})]$ is:

$\Delta E_{new} = E[-\sum_k^{z_{new}} p(z_k log(p(z_k))]$ with $p = \frac{(M-m)}{m}$

and $E[H(Z_{new}) - H(Z_{old})]$ is

$\Delta E_{coll} = E[-\sum_k^{z_{coll}} p(z_k log(p(z_k))]$ with $p = \frac{(m)}{M}$

We note that z_{new} denotes new states derived from the input and z_{coll} denotes the new states along with duplicate states as a result of the input being introduced.

Thus the expected change in entropy from one round to another can be modeled as: $p \cdot \Delta E_{new} + (1-p) \cdot \Delta E_{coll}$

After m rounds the expected change in entropy is $m \cdot (p \cdot \Delta E_{new} + (1-p) \cdot \Delta E_{coll})$.

Hence, the expected number of examples m needed such that $H(Z) < \varepsilon$ is $\frac{p \cdot \Delta E_{new} + (1-p) \cdot \Delta E_{coll})}{\varepsilon} \cdot H_{ideal}(Z)$.

Thus we can derive the expected value of m such that the final state $H(Z) < \varepsilon$.

Given the nature of the state space $|X|$, we note that m is intractable over reasonable values of ε. Hence, we present an algorithm that uses our machine learning framework to derive adversarial examples given a set of seeded known adversarial examples X_{init}.

5.4.2 Uncovering Adversarial Examples

We present an algorithm that uses the presented Markovian framework, blackbox definition $B(\cdot)$ and an initial set of known adversarial examples X_{init} to uncover a larger set of adversarial examples X_{adv}.

We present our algorithm on the next page.

We derive $x_{new} = \text{argmax}_x \sum_{i=1}^{|Z|} p(x|z_i)$ using Bayes rule.

Specifically, $p(x|z_i) = \frac{p(z_i|x) \cdot p(x)}{p(z_i)}$.

We can derive $p(z_i|x)$ via our pdf. We can derive $p(z_i)$ similarly by marginalizing out the conditional variables from our pdf. We assign a uniform prior on $p(x)$ such that $p(x) = \frac{1}{2^N}$ where N is the number of bits that represent x. This is because x represents a randomly chosen input.

We note that although 1 isnt tractable when asked to find the global set of inputs that yield low entropy states, given a set of seeded examples 1 uses the properties of the

Algorithm 1 Adversarial Alg

1: Assume we have pdfs that give $p(z_i|z_j)$, $p(z_i|x_j)$ $\forall i, j$
2: Assume as input we have $X_{init} = \{x_1, ..., x_n\}$
3: We return **M** new adversarial examples denoted in our set X_{adv}
4: Initialize X_{adv}, X_{seen} as the empty state
5: Derive the set of states Z by processing each $x_i \in X_{init}$ through $B(\cdot)$
6: While $|X_{adv}| < M$

- Derive $x_{new} = \text{argmax}_x \sum_{i=1}^{|Z|} p(x|z_i)$
- If x_{new} is not in X_{seen}, add x_{new} to X_{adv}
- Add x_{new} to X_{seen}.

derived pdfs to find locally similar adversarial examples.

This allows us to implicitly characterize a notion of similarity around the properties of adversarial states and inputs for a given hash function.

5.5 Finding Weak Hash Functions

We show how finding adversarial examples can lead us to exploit certain properties in a hash function if $B(\cdot)$ is weak. Specifically, we tie the notion of low entropic states Z to weak properties of hash functions. Interesting weaknesses of hash functions are as follows.

- Collisions - It should be computationally hard to find two inputs x_i and x_j, such that the output of the hash function yields the same result.
- Correlations - A property of cryptographic hash functions such that for small changes in inputs, the outputs should appear uncorrelated. A non-uniform distribution of the outputs that is correlated in some way with the distribution of the inputs, yields correlated inputs.
- Preimage - Preimage resistance is the property of a hash function that it is hard to invert, that is, given an element in the range of a hash function, it should be computationally infeasible to find an input that maps to that element.

In the following section we explore how given a low entropy set of states Z_{adv} and the corresponding input X_{adv}, we can uncover collisions. We present the algorithm and provide theoretical bounds on the expected number of examples we need in X_{adv} to uncover a collision. Our theoretical bounds are derived as a direct function of the entropy of the states in Z_{adv}.

For brevity, we will neglect to present specific algorithms for deriving adversarial examples that breach the preimage and correlation properties, but note that such algorithms could be developed within the reduced entropy framework.

5.5.1 Collisions

We first present our algorithm for finding collisions. We assume we have a set of inputs X_{adv} such that $H(Z_{adv}) < K \cdot \varepsilon$ where $K = H(Z_{ideal})$.

Algorithm 2 Collisions Alg

1: Assume as input X_{adv}, Z_{adv} and the hash function $B(\cdot)$
2: Initialize $Z_{seen} = \{\}$, num-col = 0
3: **for** each x_i in X_{adv}

- compute the corresponding hash $B(x_i) = z_i$
- If $z_j \in Z_{seen}$:
- num-col=num-col+1

4: return num-col

Our algorithm presented above simply loops through the final states of each x_i and computes the number of collisions. A key part of our algorithmic pipeline is we can directly link the entropy of our output states $H(Z_{adv})$ to the notion of collisions. This allows us to tie the expected bounds of our algorithms for finding adversarial examples directly to the probability of finding a collision.

We present Theorem 2 below which outlines the probability of finding a collision as a function of the entropy of the states in Z_{adv}.

Theorem 2 *Given an ε-reduction in the entropy of states (where we define it to be such that $\frac{H(Z_{adv})}{H(Z_{ideal})} < \varepsilon$), the probability of a collision in the final states is $\geq f(\varepsilon)$.*

We derive $f(\varepsilon)$ and give the proof below:

More rigorously, given X_{adv}, assume we have $\{x_i^n, z_{ij}^n\}$ pairs such that $H(Z) < K \cdot \varepsilon$ where $K = H(Z_{ideal})$, Z is the set of states indexed over i, j, n. (note i indexes over the chunks of size I, j indexes over the set of states for a given chunk of size J, n indexes over the set of inputs in X_{adv} of size N). We derive the probability that two elements in Z_{Final} collide where Z_{Final} denotes a subset of Z that contains the final states (more specifically a given element in Z_{final} is the output of the hash function given an input x^n). Note that the size of our set Z_{final} is N

We first derive P(two states in Z collide | $H(Z) < K \cdot \varepsilon) = P_A$. We then find P(two states in Z_{final} collide | two states in Z collide) $= P_B$. We note that P(two states in Z_{final} collide | $H(Z) < K \cdot \varepsilon) = P_A \cdot P_B$.

We derive a lower bound on P_A as follows.

Let us denote P(atleast two states collide) as $P(A)$. We want to derive $P(A)$ as a function of $K \cdot \varepsilon$.

$P(A) = 1 - \prod_{k=1}^{Z}(1 - \sum_{j=1}^{k} p(z_j))$ where $\prod_{k=1}^{Z}(1 - \sum_{j=1}^{k} p(z_j))$ represents the probability that no states collide.

We are given that $\sum_{j=1}^{|Z|} -p(z_j) \cdot log p(z_j) < K \cdot \varepsilon$.

It can be shown that $(1 - \sum_{j=1}^{k} p(z_j)) < \sum_{j=1}^{|Z|} -p(z_j) \cdot log p(z_j)$ when the entropy $H(Z)$ is below a threshold.

This implies $\prod_{k=1}^{|Z|}(1 - \sum_{j=1}^{k} p(z_j)) < \prod_{k=1}^{|Z|} K \cdot \varepsilon since \prod_{k=1}^{|Z|} K \cdot \varepsilon > 0$

Hence we have $1 - \prod_{k=1}^{|Z|}(1 - \sum_{j=1}^{k} p(z_j) \geq 1 - \prod_{k=1}^{|Z|} K \cdot \varepsilon$

which implies $P_A \geq 1 - \prod_{k=1}^{|Z|} K \cdot \varepsilon$. Thus we have bounded P_A from below as a function of ε.

We derive P_B as follows. We wish to find the probability that two states in Z_{final} collide given that two states in Z collide. We note that $|Z_{final}| = N$ and $|Z| = I \cdot J \cdot N$. We also assume that $|Z_{final}|$ is a uniformly sampled subset of Z for the purposes of this proof. Hence, we can view this as a problem of sampling without replacement. We note that $P(B) = \frac{N \cdot (N-1)}{(I \cdot J \cdot N) \cdot (I \cdot J \cdot N) - 1}$.

We also note that if there is more than one collision in Z, the probability of atleast one collision goes up. For the purposes of our proof, we bound $P(B) \geq \frac{N \cdot (N-1)}{(I \cdot J \cdot N) \cdot (I \cdot J \cdot N) - 1}$.

Hence, we see that the probability of a collision in the final states is $P_A \cdot P_B \geq f(\varepsilon)$ where $f(\varepsilon) = (1 - \prod_{k=1}^{|Z|} K \cdot \varepsilon) \cdot \frac{N \cdot (N-1)}{(I \cdot J \cdot N) \cdot (I \cdot J \cdot N) - 1}$.

Thus we have established a lower bound on the probability of a collision as a function of the entropy.

5.6 Validating Adversarial Examples

There are a number of known attacks against MD5 and other hash functions. Wang, et. al [35] introduced the two block attack against MD5. This attack changes two consecutive 512 bit chunks of the input such that the state output of the first block nearly collides with that of the unmodified message and the second block collides with the same. Wang's technique for breaking MD5 introduces the notion of multi-message modification. The multi-message modification technique increases the tractability of discovering input blocks that meet the sufficient conditions for MD5 and leverage weaknesses in the function compositions of MD5. Similarly exploiting weaknesses in the internal state transitions in MD5, Klima [17] discovered a way to decrease the search space for MD5. These attacks and others which find collisions in hash functions are all premised on uncovering a set of sufficient conditions that any inputs that collide must meet.These collisions are always a function of the implementation of the particular function as outlined in Section 5.1.

We note that domain based attacks mostly look to exploit weak function compositions in hash functions by finding adversarial examples that uncover such weaknesses. Our adversarial learning framework aims to achieve the same end through empirical means. Our blackbox construction maintains a model of the Merkle–Damgård functions that is specific enough to capture the characteristics of function compositions. Yet we dont assume anything about the function compositions themselves and instead use our trained probability distribution functions to posit relationships between inputs and states. Thus, our adversarial learning framework takes a structured approach to finding relationships between adversarial examples and weak function compositions across the class of Merkle–Damgård functions. Thereby, we converge to the same principles of the domain based approaches while maintaining the abstractness of the framework model.

6 Results & Discussion

In this section we outline our experiments, present the results and analyze them.

6.1 Experiments

We design experiments to observe parameter values of Theorem 1 and Theorem 2 in practice. Specifically, we empirically validate the result of Theorem 1, regarding the expected number of examples needed before we can expect the entropy of the states $H(Z) < H(Z_{ideal}) \cdot \varepsilon$. We also use the machine learning construction of Theorem 2 to derive new inputs that yield collisions with high probability given a set of seeded inputs. Our empirical results for Theorem 2 illustrate the relationship between states and inputs and shows how new adversarial examples can be generated from a set of seeds based on trained probability distribution functions.

We experiment exclusively on MD5 for ease of validation given the scale of its' known weaknesses, noting that our methodology is equally applicable to SHA1 and SHA2. We apply our framework to MD5 given the reduction in Section 5.2.1. All of our experiments were run on a box with 16 2.3 GHz Intel Xeon E5-2686 v4 Processors and 122 GiB RAM.

To observe parameters of Theorem 1 for MD5, we compute values of m as a function of ε for multiple input spaces X. In addition to running this experiment on all states observed during the computation of MD5, we also narrow our focus to states observed by a particular blackbox component,

in the case of MD5 being B^0, B^1, B^2, B^3, mapping to MD5 rounds F, G, H, I, respectively.

To observe parameters of Theorem 2, we leverage Algorithms 5.4.2 and 5.5.1 in order to find collisions in the predefined input spaces X, also observing results for all states observed during the MD5 computation as well as for the individual blackbox components.

We ran each of these experiments on six pre computed input spaces denoted random_$\{0,1,2\}$ and seeded_$\{0,1,2\}$, respectively. Each set from random_$\{0,1,2\}$ was contains 200,000 inputs, x of 1024 randomly generated bits using Java's built in `randint()` function. In addition to the input blocks, the set contains the states observed during the computation of MD5(x), for each x in the set as well as the states observed, as defined in Section 5.2 during the computation of the hash value, including the final state which is the output of the hash function. seeded_$\{0,1,2\}$ was generated the same way, except that the input blocks x had the first 512 bits fixed and the second 512 bits generated randomly. The first 512 bits were fixed using the preface included by Wang in [35], and noting her observation that given a sufficient 512 bit first block, the next 512 blocks will force a collision with increased likelihood. These sets were used to bring the computational requirements of the problem into the realm of our constrained computational resources, though we note that this has no impact on the theoretical guarantees outlined in the previous sections.

6.2 Experimental Results and Discussion

6.2.1 Theorem 1

In our Theorem 1 experiments we observe how many samples must be taken from each input space before a reduction in the entropy of the observed states is reduced, that is that there is a collision of two observed states. Taking Wang, et. al's claim to be true, we expect that the seeded input spaces will in many cases yield a reduction in entropy at some point, whilst seldom seeing a reduction in entropy in the state spaces generated fully randomly.

We observe this claim holds in our framework in Figure 2, which observes duplicated states over all blackbox components $B^{0,1,2,3}$. Here we see a reduction in entropy of each of the seeded input sets, demonstrated by the reduced m values that occur in each of these series. As expected, we also observe that the entropy of the random sets is never reduced, even after exhausting the full input space, which is observed in that the random series in Figure 2 never show a reduction in entropy even after all M possible input states are observed.

From Figure 2 we can also observe the expected behavior of m decreasing for larger values of ε. When the ε tolerance for a sufficient entropy reduction was $\geq .65$, each of the seeded input sets showed a maximum reduction in entropy for small values m. The reduced entropies in the seeded sets before this ε value can be attributed to the fact that the mth input value x is selected at random from the input space, treated as a uniform distrobution. It is therefore expected that for small values of ε the m value will in some cases be small.

In Figures 3, 4, 5, 6 we observe the same, but within the context of each particular blackbox component. In the case of B^0, we observe in Figure 3 the same trend holds wherein each of the seeded sets experience reduced entropy when $\varepsilon \geq 0.5$, and additionally that there is never an observed reduced entropy for the randomly generated sets. The same is observed for component B^2 in Figure 5.

B^1 and B^3 are the blackbox components that never experiences a reduced entropy. In 4 and 6 we observe that there is no reduced entropy caused by any of the input spaces. In each of these cases, the all of the elements from the input space are selected without a reduced entropy every being observed, signifying that there are also no repeated states. These blackbox components map to the G and I rounds in MD5, respectively, suggesting they increase collision resistance over the other rounds.

We observe this claim holds in our framework in Figure 2, which observes duplicated states over all blackbox components $B^{0,1,2,3}$. Here we see a reduction in entropy of each of the seeded input sets, demonstrated by the reduced m values that occur in each of these series. As expected, we also observe that the entropy of the random sets is never reduced, even after exhausting the full input space, which is observed in that the random series in Figure 2 never show a reduction in entropy even after all M possible input states are observed.

From Figure 2 we can also observe the expected behavior of m decreasing for larger values of ε. When the ε tolerance for a sufficient entropy reduction was $\geq .65$, each of the seeded input sets showed a maximum reduction in entropy for small values m. The reduced entropies in the seeded sets before this ε value can be attributed to the fact that the mth input value x is selected at random from the input space, treated as a uniform distrobution. It is therefore expected that for small values of ε the m value will in some cases be small.

In Figures 3, 4, 5, 6 we observe the same, but within the context of each particular blackbox component. In the case of B^0, we observe in Figure 3 the same trend holds wherein each of the seeded sets experience reduced entropy when $\varepsilon \geq 0.5$, and additionally that there is never an observed reduced entropy for the randomly generated sets. The same is

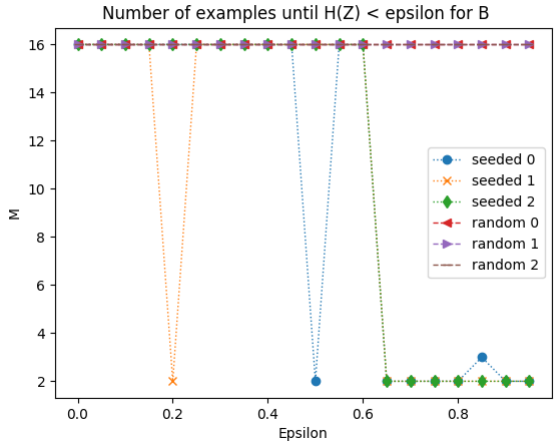


Figure 2: Number of examples m to drive entropy over all of the blackbox components $B^{0,1,2,3}$ below $\kappa \cdot \varepsilon$ for $\varepsilon \in \{0, .05, .10, .15..., 1\}$

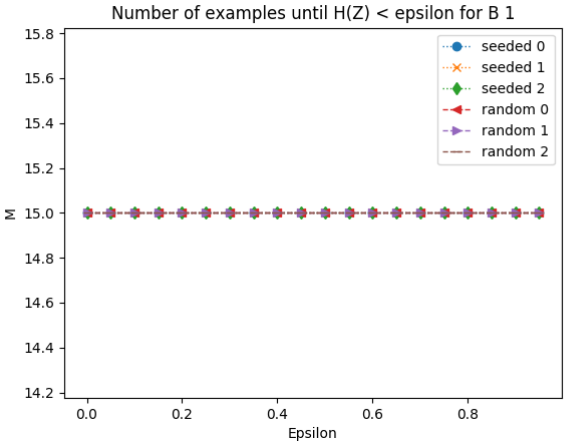


Figure 4: Number of examples m to drive entropy over all of the blackbox components B^{1} below $\kappa \cdot \varepsilon$ for $\varepsilon \in \{0, .05, .10, .15..., 1\}$

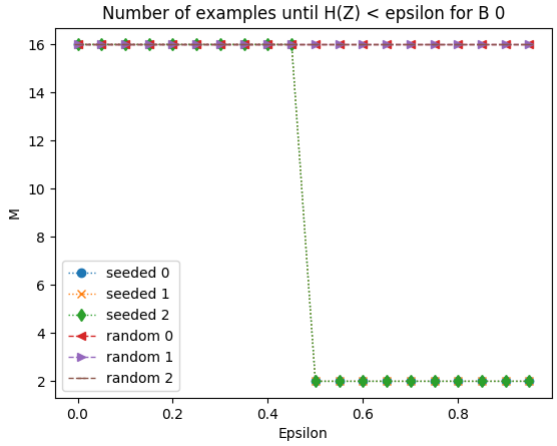


Figure 3: Number of examples m to drive entropy over all of the blackbox components B^{0} below $\kappa \cdot \varepsilon$ for $\varepsilon \in \{0, .05, .10, .15..., 1\}$

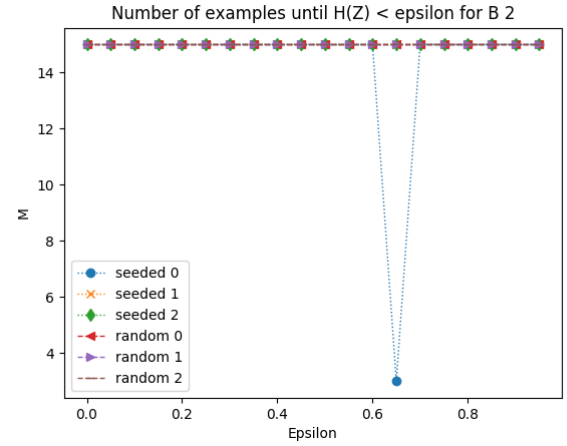


Figure 5: Number of examples m to drive entropy over all of the blackbox components B^{2} below $\kappa \cdot \varepsilon$ for $\varepsilon \in \{0, .05, .10, .15..., 1\}$

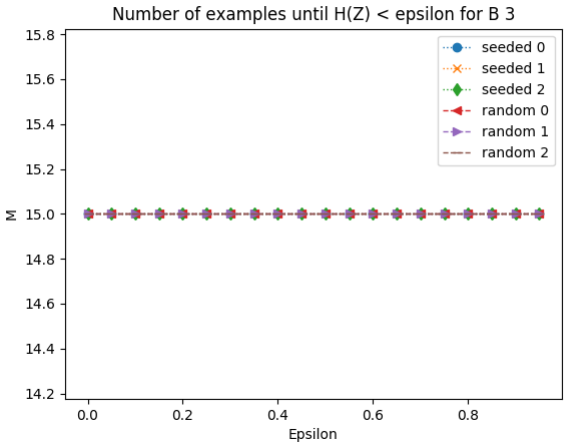


Figure 6: Number of examples m to drive entropy over all of the blackbox components B^3 below $\kappa \cdot \varepsilon$ for $\varepsilon \in \{0, .05, .10, .15..., 1\}$

observed for component B^2 in Figure 5.

B^1 and B^3 are the blackbox components that never experiences a reduced entropy. In 4 and 6 we observe that there is no reduced entropy caused by any of the input spaces. In each of these cases, the all of the elements from the input space are selected without a reduced entropy every being observed, signifying that there are also no repeated states. These blackbox components map to the G and I rounds in MD5, respectively, suggesting they increase collision resistance over the other rounds.

6.2.2 Theorem 2

In our experiments for Theorem 2, we maximize the amount of state collisions that occur, and in doing so maximize the reduction in entropy for a given number of samples m from the input space.

In Figure 7 we observe the number of state collisions observed as the number of samples m taken from the input space increases. There are no collisions found from and of the random spaces. This is the expected behavior over a small random sample. It would be a very weak hash function that experienced state collisions over a small random space. However, our algorithm was able to find collisions in the seeded input spaces even for small values of m. We observe that with values as small as $m = 5$, our adversarial algorithm uncovers state collisions in the seeded input spaces.

Figure 8 shows the entropy as a function of m for each input space. The expected relationship between entropy and sample size is that the entropy increases as the number of obsereved states in the sample increases. This is the relationship that is observed in 8. We would additionally expect to see, in the general case, that the entropy of the seeded sets would be ε less than that of the random sets. This is the result for seeded_0 and seeded_1, however this is not the case for seeded_2. This is so for the same reason it was in the Theorem 1 results in that with randomly selected inputs from a fixed input space there is a nonnegligable probability that the same element from the random input space is read more than once.

7 Future Work

The primary purpose of this paper is to present a new paradigm for finding weakness in cryptographic systems. Thus, there is much future work that can be carried out in applying this framework, and better applying this framework to other domains. There are three types of work to be pursued following this paper. These directions are, respectively, (1) exploring other types of weaknesses, (2) applying the methodology to other hash functions and cryptographic systems, and (3) running the presented algorithms on a larger state space. Each of these directions will be taken up in turn in this section.

First, additional algorithms can be developed to leverage the framework presented in this paper towards discovering other types of weaknesses in hash functions. In addition to being collision resistant, cryptographic hash functions should be resistant to preimage and correlation attacks. The notion of discovering states that are susceptible to collisions can be used to reduce the work load on a SAT solver attempting a preimage attack by reducing the amount of unknown intermediate states. SAT solvers have been successfully applied to hash functions in similar contexts [22]. Additional algorithms could be developed for correlation attacks.

Secondly, this methodology should be applied to other hash functions and cryptographic systems. With respect to hash functions, one of the key strengths of our framework is that it can easily be applied to any hash function of the Merkle–Damgård construction. In this paper we explored applications to MD5, but more computationally expensive applications of the framework should be carried out against SHA1 and SHA2 in pursuit of winding weaknesses of these hash functions. With respect to other cryptograhpic systems, see the problems laid out in Section 4.2.

Lastly, more deliberate algorithms can be applied to efficiently traverse the input and state space iteratively. We presented a Markov model that finds similar states and inputs in a greedy and localized way. More work can be done in exploring a greater class of algorithms that performs intelligent sampling across the input space to uncover sets of adversarial examples. Furthermore, interesting seeding techniques can be explored that attempt to search in various localized spaces beyond random or deliberate seeding.

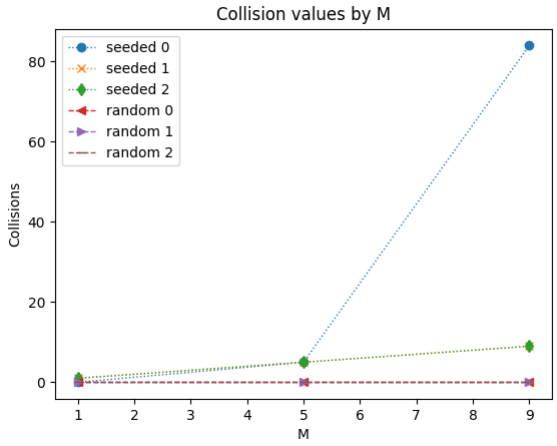


Figure 7: Number of states observed more than once, denoted collisions, over all of the blackbox components $B^{1,2,3,4}$ for a varying number of inputs, m, sampled from each input space.

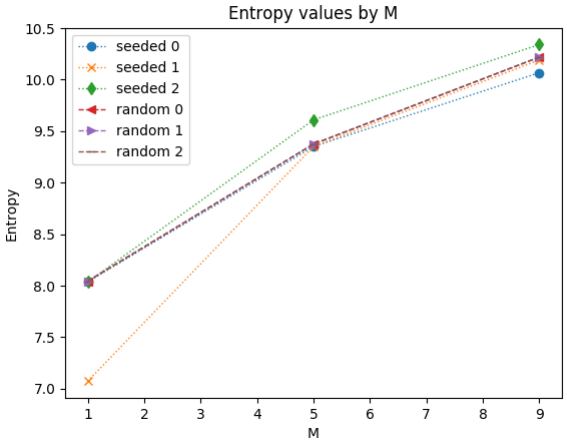


Figure 8: Entropy of the set of observed states over all of the blackbox components $B^{1,2,3,4}$ for a varying number of inputs, m, sampled from each input space.

Finally greater computational resources can be applied to find additional interesting properties of MD5. Due to resource constraints, we could only apply our framework to explore an input space of $\sim 2^{21}$. Running the experiments run in this paper on larger input spaces would yield additional interesting results about MD5.

Conclusion

In this paper we have presented a novel paradigm that considers the notion of adversarial learning in the context of cryptographic systems. We started by expanding the definition of adversarial learning beyond supervised machine learning systems. We then constructed a machine learning framework for the class of Merkle-Damgard hash functions that aims to uncover adversarial examples. We derived theoretical bounds on the expected number of examples needed before seeing a reduction in the entropy of the states. We then presented a Markov based algorithm that constructed distribution functions that related states to previous states and inputs. Finally we tied the notion of adversarial examples and entropy reduction to collisions, thus linking the number of examples to the probability of collision. We ran experiments on entropy reduction and the number of collisions using randomly seeded and deliberately seeded data for the MD5 algorithm. Thus, we showed our algorithm was able to infer new adversarial examples given a seeded set by exploiting the relationship between states and input. There is much future work that can be done in expanding the presented framework to other cryptographic systems, refining the current framework to exploit other weak properties of hash functions, and coming up with further novel algorithms for efficiently searching the state and input spaces.

Availability

The code that implements our experiments and the data used are available on Github at https://github.com/gangopad/hash-project.

References

[1] Anish Athalye, Nicholas Carlini, and David Wagner. Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples. *arXiv preprint arXiv:1802.00420*, 2018.

[2] Andrey Bogdanov, Dmitry Khovratovich, and Christian Rechberger. Biclique cryptanalysis of the full aes. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 344–371. Springer, 2011.

[3] Ivan Bjerre Damgård. A design principle for hash functions. In *Conference on the Theory and Application of Cryptology*, pages 416–427. Springer, 1989.

[4] Christophe De Canniere, Florian Mendel, and Christian Rechberger. Collisions for 70-step sha-1: on the full cost of collision search. In *International Workshop on Selected Areas in Cryptography*, pages 56–73. Springer, 2007.

[5] Christophe De Canniere and Christian Rechberger. Finding sha-1 characteristics: General results and applications. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 1–20. Springer, 2006.

[6] Bert Den Boer and Antoon Bosselaers. An attack on the last two rounds of md4. In *Annual International Cryptology Conference*, pages 194–203. Springer, 1991.

[7] Bert den Boer and Antoon Bosselaers. Collisions for the compression function of md5. In *Workshop on the Theory and Application of of Cryptographic Techniques*, pages 293–304. Springer, 1993.

[8] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[9] Thore Graepel, Kristin Lauter, and Michael Naehrig. Ml confidential: Machine learning on encrypted data. In *International Conference on Information Security and Cryptology*, pages 1–21. Springer, 2012.

[10] Zvi Gutterman, Benny Pinkas, and Tzachy Reinman. Analysis of the linux random number generator. In *Security and Privacy, 2006 IEEE Symposium on*, pages 15–pp. IEEE, 2006.

[11] Nadia Heninger and Hovav Shacham. Reconstructing rsa private keys from random key bits. In *Advances in Cryptology-CRYPTO 2009*, pages 1–17. Springer, 2009.

[12] Gabriel Hospodar, Benedikt Gierlichs, Elke De Mulder, Ingrid Verbauwhede, and Joos Vandewalle. Machine learning in side-channel analysis: a first study. *Journal of Cryptographic Engineering*, 1(4):293, 2011.

[13] Ling Huang, Anthony D Joseph, Blaine Nelson, Benjamin IP Rubinstein, and JD Tygar. Adversarial machine learning. In *Proceedings of the 4th ACM workshop on Security and artificial intelligence*, pages 43–58. ACM, 2011.

[14] Dejan Jovanović and Predrag Janičić. Logical analysis of hash functions. In *International Workshop on Frontiers of Combining Systems*, pages 200–215. Springer, 2005.

[15] Jonathan Katz and Yehuda Lindell. *Introduction to modern cryptography*. CRC press, 2014.

[16] John Kelsey, Bruce Schneier, David Wagner, and Chris Hall. Cryptanalytic attacks on pseudorandom number generators. In *International Workshop on Fast Software Encryption*, pages 168–188. Springer, 1998.

[17] Vlastimil Klima. Finding md5 collisions on a notebook pc using multi-message modifications. *IACR Cryptology ePrint Archive*, 2005:102, 2005.

[18] Vlastimil Klima. Tunnels in hash functions: Md5 collisions within a minute. *IACR Cryptology ePrint Archive*, 2006:105, 2006.

[19] Alexey Kurakin, Ian Goodfellow, and Samy Bengio. Adversarial machine learning at scale. *arXiv preprint arXiv:1611.01236*, 2016.

[20] Jie Liang and Xue-Jia Lai. Improved collision attack on hash function md5. *Journal of Computer Science and Technology*, 22(1):79–87, 2007.

[21] Ralph Merkle. Secrecy, authentication, and public key systems. *Ph. D. Thesis, Stanford University*, 1979.

[22] Ilya Mironov and Lintao Zhang. Applications of sat solvers to cryptanalysis of hash functions. In *International Conference on Theory and Applications of Satisfiability Testing*, pages 102–115. Springer, 2006.

[23] Nicolas Papernot, Patrick McDaniel, Ian Goodfellow, Somesh Jha, Z Berkay Celik, and Ananthram Swami. Practical black-box attacks against machine learning. In *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*, pages 506–519. ACM, 2017.

[24] Ronald L Rivest. Cryptography and machine learning. In *International Conference on the Theory and Application of Cryptology*, pages 427–439. Springer, 1991.

[25] Yu Sasaki, Yusuke Naito, Jun Yajima, Takeshi Shimoyama, Noboru Kunihiro, and Kazuo Ohta. How to construct sufficient conditions for hash functions. In *Progress in Cryptology-VIETCRYPT 2006*, pages 243–259. Springer, 2006.

[26] Bruce Schneier. Cryptanalysis of md5 and sha: Time for a new standard. 2004.

[27] Marc Stevens, Elie Bursztein, Pierre Karpman, Ange Albertini, and Yarik Markov. The first collision for full sha-1. In *Annual International Cryptology Conference*, pages 570–596. Springer, 2017.

[28] Marc Stevens and Daniel Shumow. Speeding up detection of sha-1 collision attacks using unavoidable attack conditions. *IACR Cryptology ePrint Archive*, 2017:173, 2017.

[29] Marc Stevens, Alexander Sotirov, Jacob Appelbaum, Arjen Lenstra, David Molnar, Dag Arne Osvik, and Benne De Weger. Short chosen-prefix collisions for md5 and the creation of a rogue ca certificate. In *Advances in Cryptology-CRYPTO 2009*, pages 55–69. Springer, 2009.

[30] Marc Martinus Jacobus Stevens et al. *Attacks on hash functions and applications*. Mathematical Institute, Faculty of Science, Leiden University, 2012.

[31] Jiawei Su, Danilo Vasconcellos Vargas, and Kouichi Sakurai. One pixel attack for fooling deep neural networks. *IEEE Transactions on Evolutionary Computation*, 2019.

[32] Florian Tramèr, Alexey Kurakin, Nicolas Papernot, Ian Goodfellow, Dan Boneh, and Patrick McDaniel. Ensemble adversarial training: Attacks and defenses. *arXiv preprint arXiv:1705.07204*, 2017.

[33] Xiaoyun Wang, Xuejia Lai, Dengguo Feng, Hui Chen, and Xiuyuan Yu. Cryptanalysis of the hash functions md4 and ripemd. In *Annual international conference on the theory and applications of cryptographic techniques*, pages 1–18. Springer, 2005.

[34] Xiaoyun Wang, Yiqun Lisa Yin, and Hongbo Yu. Finding collisions in the full sha-1. In *Annual international cryptology conference*, pages 17–36. Springer, 2005.

[35] Xiaoyun Wang and Hongbo Yu. How to break md5 and other hash functions. In *Annual international conference on the theory and applications of cryptographic techniques*, pages 19–35. Springer, 2005.

[36] Xiaoyun Wang, Hongbo Yu, and Yiqun Lisa Yin. Efficient collision search attacks on sha-0. In *Annual International Cryptology Conference*, pages 1–16. Springer, 2005.

[37] Scott Yilek, Eric Rescorla, Hovav Shacham, Brandon Enright, and Stefan Savage. When private keys are public: results from the 2008 debian openssl vulnerability. In *Proceedings of the 9th ACM SIGCOMM Conference on Internet Measurement*, pages 15–27. ACM, 2009.