# Eve's SHA3 candidate: malicious hashing

Jean-Philippe Aumasson*

Nagravision SA, Switzerland

**Abstract.** We investigate the definition and construction of hash functions that incorporate a backdoor allowing their designer (and only her) to efficiently compute collisions, preimages, or more. We propose semi-formal definitions of various types of malicious generators—i.e. probabilistic algorithms modeling a malicious designer—and of the intuitive notions of undetectability and undiscoverability. We describe relations between the notions defined as well as basic strategies to design malicious hashes. Based on the observation that a backdoor can be at least as hard to discover as to break the underlying hash, we present a backdoored version of the SHA3 finalist BLAKE. This preliminary work leaves many open points and challenges, such as the problem of finding the most appropriate definitions. We believe that a better understanding of malicious uses of cryptography will assist combat it; malicious hash functions are indeed powerful tools to perform insider attacks, government espionnage, or software piracy.

**Keywords**: hash functions, backdoors, malicious cryptography, cryptovirology

## 1 Introduction

In the 1983 movie *Wargame*, a programmer of a military supercomputer hides the feature that logging on with his son's name as a password gives access to undocumented parts of the system, which in the movie almost leads to a nuclear war—this is an example of *application backdoor*. A nonfiction example is the 2003 attempt to backdoor the Linux 2.6 kernel [1], when one added three lines of code in `kernel/exit.c` to allow illegitimate root access. The maintainers quickly noticed, for the culprit directly modified the CVS tree rather than committing a change with the usual mechanism. More recently, the FBI was accused of planting application backdoors in the IPSEC stack of OpenBSD[1].

In the same spirit as application backdoors, *cryptographic backdoors* are intentionally designed weaknesses in a cryptographic algorithm that allow the designer to obtain information, or to perform a computation, that is supposed to be infeasible for a normal user. For example, Shumow and Ferguson discovered [2] the possibility of a backdoor in the NSA-designed `Dual_EC` algorithm, a NIST-standard pseudorandom generator [3] whose security relies on a hard elliptic-curve problem. The alleged backdoor consists in the knowledge of the exponent $e$ mapping the specified constant point $Q$ to the generator $P$; knowing $e$ would allow one to determine the internal state of the pseudorandom generator, and thus to predict all future outputs (more problems with `Dual_EC` were reported in [4, 5]).

Other famous suspicions of cryptographic backdoors concerned the S-boxes of DES [6, Ch.9] [7, §12.3] and the encryption devices by the Swiss company Crypto AG [8], though the former allegation is now known to be unfounded[2]. Other cases that may be considered as backdoors are the intentionally weak algorithms allowed for export in the 90's, such as A5/2 in GSM technology, or the password protection in Lotus' Ami Pro word processor.

*Part of this work was done while the author was with FHNW, Switzerland.

[1]See `http://marc.info/?l=openbsd-tech&m=129236621626462&w=2` (Accessed 21/01/2011).

[2]Cf. NSA's Dickie George's comments during the Cryptographers Panel at the RSA Conference 2011.

Cryptographic backdoors can also be found in *implementations*. For instance, Wagner and Biondi [9] discovered how to plant a simple and powerful backdoor in a C implementation of RC4. Hardware implementations are not immune to cryptographic backdoors, as shown by bug attacks [10].

After mentioning backdoors in pseudorandom generators, block ciphers, and stream ciphers, this paper proposes to investigate backdoors in *cryptographic hash functions*, a topic yet unstudied in the published literature. Hash functions incorporating a backdoor[3] will henceforth be called *malicious hash functions*.

We believe that cryptographic backdoors should be considered as a realistic threat, and that their study should not be left to government agencies and to malevolent parties. A better understanding of the (im)possibilities of an adversary to use cryptography maliciously will help combat it, as previously argued by Young and Yung [11], who coined the term "cryptovirology" to denote this field of study.

## 1.1 Applications of malicious hash functions

Like application backdoors most applications of crypto backdoors are malicious, though one could imagine less malicious ones such as DRM's. Be it in software or in cryptography, backdoors lead to powerful attacks and can be very hard to detect, as they typically bypass most security countermeasures and are most easily planted by insiders[4]. Furthermore, backdoors can potentially affect many systems with little effort. To illustrate these properties, below are some simplified scenarios wherein an ill-intentioned individual or institution could exploit a backdoor in a hash function:

- Eve has to design the hash function of PONY's GameStation 3 console for authenticating boot code and games. To kill two birds with one stone, she creates a hash for which she can efficiently compute second preimages. This will allow her to
  1. Execute arbitrary code on her GS3, including pirated games and malware;
  2. Anonymously blackmail PONY by presenting evidence of a weakness in the authentication mechanism.

  Of course, the latter should be done only if the evidence does not reveal the presence of the backdoor (suggesting the notion of *deniable backdoors*, which can be related to the notion of deniable password snatching in [11, p97]). This illustrates the power of what we shall call *dynamic second preimage backdoors*.
- The intelligence agency of Freedonia assists a foreign cryptography company to improve the resistance to tampering and reverse engineering of its new secure chipset, which is to be used by thousands of machines of the Lower Slobbovia government. Besides completing its mandate, the agency surreptitiously modifies the hardware, for "test purposes", such that a magic suffix $s$ exists for which $H(m\|s)$ returns some encoding of $m$'s first bits. This will allow the agency to recover HMAC keys with a single chosen-message query. Such malicious hardware implementations of hash functions are briefly discussed in §§3.1.
- Charlie is the submitter of the SHA3 candidate Bløccakein, and designed it such that he knows two (and only two) colliding messages $m$ and $m'$, but such that the function is

[3]Note that, in cryptography, a backdoor differs from a trapdoor in that the former is supposed to be hidden, whereas the latter is a publicly know property of, say, a one-way permutation. Trapdoor hash functions have previously been defined, e.g. VSH.

[4]For a thorough study of the insider threat, we refer the reader to the work by Bishop et al. [12].

otherwise secure. A few years after Bløccakein is chosen as SHA3, Charlie triggers cryptoarmageddon after anonymously publishing $m$ and $m'$ on the IACR ePrint server with claims that SHA3's broken. The type of backdoor used is what we shall call a *static collision backdoor*.

One can imagine many other applications of backdoors in hash functions.

## 1.2 Previous works

Academic research paid little attention to cryptographic backdoors and to malicious cryptography in general. Below we present some of the most relevant related works:

- In 1997 Rijmen and Preneel [13] proposed to hide linear relations in S-boxes and presented "backdoor versions" tCAST and tLOKI; these were broken in [14] along with the general strategy proposed. Rijmen and Preneel already noted that "[besides] the obvious use by government agencies to catch dangerous terrorists and drug dealers, trapdoor block ciphers can also be used for public key cryptography." [13]; indeed, [15] previously argued that a backdoor block cipher is equivalent to a public key cryptosystem.
- Still in 1997, Patarin and Goubin [16] proposed an S-box based asymmetric scheme which works by publishing the explicit equations describing a 2-round SPN, while keeping the actual S-boxes and linear transforms secret; this was broken independently by Ye et al. and Biham [17, 18].
- More recently, Young and Yung designed backdoor *blackbox* malicious ciphers, which assume that the algorithm is not known to an adversary. Such ciphers exploit low-entropy plaintexts to embed information about the key in ciphertexts [19, 20]. Young and Yung coined the term *cryptovirology* [11] and proposed various malicious applications of cryptography: ransomware, deniable data stealing, etc.
- Bug attacks by Biham, Carmeli, and Shamir [10] are implementation backdoors for cryptographic algorithm, via the introduction of a bug in the hardware; no backdoor is hidden in the actual algorithm, but an attacker exploits a bug in the multiplication instruction of the processor to attack a class of cryptographic algorithms. A related topic is that of *hardware trojans* (design and detection).
- In 2010 Filiol [21] proposed to use malicious pseudorandom generators to assist the creation of executable code difficult to reverse-engineer with static and dynamic analysis. Typical applications are the design of malwares that resist detection methods that search for what looks like obfuscated code (suggesting the hiding of malicious instructions).

For a good overview of backdoors we refer to Wysopal and Eng's 2007 paper [22], which categorizes computer backdoors in three types:

1. *Crypto backdoors*, designed weaknesses for a particular key or message.
2. *System backdoors*: malware written to compromise a system, sometimes relying on social engineering for initial execution.
3. *Application backdoors*: modifications to legitimate programs designed to bypass security mechanisms.

As [22] focuses on application backdoors, it refines it definition by describing four classes of application backdoors: special credentials, hidden functionality, unintended network activity, and manipulation of security critical parameters.

### 1.3 Contribution and future work

We propose semi-formal definitions of malicious hash functions as *adversaries* composed of a pair of algorithms: a (probabilistic) *malicious generator* and an *exploit algorithm*. Based on this formalism, we define intuitive notions of undetectability and undiscoverability. Relations between those notions are discussed as well as examples of strategies to design concrete malicious hash functions. We introduce dedicated notions as *neutral structure*, *entropy spraying*, and *chameleon structure*. As a proof of concept, a maliciously modified version of the SHA3 finalist BLAKE is proposed: using simple techniques, we build a version of BLAKE for which we can find arbitrarily many collisions, while ensuring that the modified version is at least as secure as the original BLAKE.

To the best of our knowledge, no research on malicious hash functions has been published. Our work is preliminary and leaves many questions and problem open, such as that of finding the most appropriate definitions. Goals of this article are to foster research in the area of cryptographic backdoors (a.k.a. cryptovirology [11]), and to assist detection and prevention of malicious cryptography. Future research may also address related problems, like that of constructing trapdoor(less) one-way *permutations* with only efficient operations (i.e. no bignum arithmetic, low memory and computation).

## 2 Defining malicious hash functions

We define several types of malicious hash functions as well as the related security notions. Contrary to typical security definitions, our adversary does not attack an existing algorithm, but rather creates the primitive, knows the secret (i.e. the backdoor and how to exploit it), while honest parties play the attackers and cryptanalyze Eve's design. Definitions should thus identify malicious hash functions with the adversary, who is then seen as a *generator* of malicious hash functions equipped with the ability to exploit a backdoor. Malicious hash functions will thus be defined by two efficient algorithms:

- A *malicious generator*, i.e. a probabilistic algorithm returning a hash function and a backdoor;
- An *exploit algorithm*, i.e. a deterministic or probabilistic algorithm that uses the knowledge of the backdoor to bypass some security property of the hash function.

We shall distinguish two types of backdoors: *static (backdoor) adversary*, which have a deterministic exploit algorithm, and *dynamic (backdoor) adversaries*, which have a probabilistic one.

Our definitions are only semi-formal, i.e. not well-defined mathematically, for example due to unspecified distributions or to being subject to arbitrary interpretations (for distinguishers). Reasons are that we failed to find definitions that are both rigorous and that capture well real-world attack scenarios. As our work is more practice- than theory-oriented, we prefered more general and flexible definitions, rather than unusable and limited mathematically sound ones. That said, we do not claim that satisfying both goals is impossible.

Below, the hash algorithms and backdoors returned as outputs of a malicious generator are assumed to be encoded as bitstring in some normal form, and to be of reasonable finite length. The generator and exploit algorithms, as well as the backdoor string, are kept secret by the malicious designer.

## 2.1 Static backdoors adversaries

We first define *static collision adversary* ($\mathsf{SCA}$). Informally, Eve is an $\mathsf{SCA}$ if she designs a hash function for which she knows one pair of colliding messages.

**Definition 1** ($\mathsf{SCA}$)**.** *A* static collision adversary *is a pair* $(\mathsf{GenSC}, \mathsf{ExpSC})$ *such that*

- *The* malicious generator $\mathsf{GenSC}$ *is a probabilistic algorithm that returns a pair* $(H, b)$*, where* $H$ *is a hash function and* $b$ *is a backdoor.*
- *The* exploit algorithm $\mathsf{ExpSC}$ *is a deterministic algorithm that takes a hash function* $H$ *and a backdoor* $b$ *and that returns distinct* $m$ *and* $m'$ *such that* $H(m) = H(m')$*.*

This definition can be generalized to define an adversary able to produce a small number of collisions, through the definition of several $\mathsf{ExpSC}$ algorithms (e.g. $\mathsf{ExpSC}_1, \ldots, \mathsf{ExpSC}_n$).

As a notion of static second preimage adversary would not concretely differ from that of static collision, our next definition relates to (first) preimages:

**Definition 2** ($\mathsf{SPA}$)**.** *A* static preimage adversary *is a pair* $(\mathsf{GenSP}, \mathsf{ExpSP})$ *such that*

- *The* malicious generator $\mathsf{GenSP}$ *is a probabilistic algorithm that returns a pair* $(H, b)$*, where* $H$ *is a hash function and* $b$ *is a backdoor.*
- *The* exploit algorithm $\mathsf{ExpSP}$ *is a deterministic algorithm that takes a hash function* $H$ *and a backdoor* $b$ *and that returns* $m$ *such that* $H(m)$ *has low entropy.*

In the above definition "low entropy" is undefined. Very informally, a low-entropy digest is one that will convince a third party that "something is going wrong" with the hash function; for example, the all-zero digest, one will all bytes identical, etc. A formal definition would require a definition of the (negligibly small) set of low entropy digests. We believe that such a definition is unnecessary, for it would necessarily rely on arbitrary choices, and for it is sufficient for low-entropy digests to pass a duck test (a.k.a. "I know it when I see it").

We now give the definition of a distinguishing backdoor, which generalizes the two previous ones:

**Definition 3** ($\mathsf{SDA}$)**.** *A* static distinguishing adversary *for a relation* $\mathcal{R}$ *is a pair* $(\mathsf{GenSD}, \mathsf{ExpSD})$ *such that*

- *The* malicious generator $\mathsf{GenSD}$ *is a probabilistic algorithm that returns a pair* $(H, b)$*, where* $H$ *is a hash function and* $b$ *is a backdoor.*
- *The* exploit algorithm $\mathsf{ExpSD}$ *is a deterministic algorithm that takes a hash function* $H$ *and a backdoor* $b$ *and that returns a list* $m_1, \ldots, m_N$ *such that*

$$\mathcal{R}\left(m_1, \ldots, m_N, H(m_1), \ldots, H(m_N)\right) \ .$$

Clearly, the relation $\mathcal{R}$ must be difficult to satisfy for an ideal function, so that the backdoor is meaningful. $\mathcal{R}$ can for example model multi-collisions, input/output linear relations, etc. $\mathsf{SCA}$ and $\mathsf{SPA}$ can thus be seen as particular cases of a static distinguishing adversary. Another requirement is that the relation should be *convincing*: for example, one can hash a random message with SHA512 and arguing that finding a preimage of the digest obtained is difficult for an ideal function, in order to claim that this gives a distinguisher for SHA512.

## 2.2 Dynamic backdoors

Dynamic backdoors naturally extend static backdoors from one or a few successful attacks to an arbitrary number. In some sense, dynamic backdoors are to static backdoors what universal forgery is to existential and selective forgery for MAC's.

**Definition 4** (DCA)**.** *A* dynamic collision adversary *is a pair* (GenDC, ExpDC) *such that*

- *The* malicious generator GenDC *is a probabilistic algorithm that returns a pair* $(H, b)$*, where* $H$ *is a hash function and* $b$ *is a backdoor.*
- *The* exploit algorithm ExpDC *is a probabilistic algorithm that takes a hash function* $H$ *and a backdoor* $b$ *and that returns distinct* $m$ *and* $m'$ *such that* $H(m) = H(m')$*.*

In this definition, the ExpDC algorithm should be seen as an efficient sampling algorithm choosing the pair $(m, m')$ within a large set of colliding pairs, as implicitly defined by GenDC. The latter may be created in such a way that sampled messages satisfy a particular property, e.g. have a common prefix.

The definitions of second preimage and preimage dynamic backdoor adversaries follow naturally:

**Definition 5** (DSPA)**.** *A* dynamic second preimage adversary *is a pair* (GenDSP, ExpDSP) *such that*

- *The* malicious generator GenDSP *is a probabilistic algorithm that returns a pair* $(H, b)$*, where* $H$ *is a hash function and* $b$ *is a backdoor.*
- *The* exploit algorithm ExpDSP *is a probabilistic algorithm that takes a hash function* $H$*, a backdoor* $b$*, and a message* $m$ *and that returns an* $m'$ *distinct from* $m$ *such that* $H(m) = H(m')$*.*

**Definition 6** (DPA)**.** *A* dynamic preimage adversary *is a pair* (GenDP, ExpDP) *such that*

- *The* malicious generator GenDP *is a probabilistic algorithm that returns a pair* $(H, b)$*, where* $H$ *is a hash function and* $b$ *is a backdoor.*
- *The* exploit algorithm ExpDP *is a probabilistic algorithm that takes a hash function* $H$*, a backdoor* $b$*, and a digest* $d$ *and that returns* $m$ *such that* $H(m) = d$*.*

In the definitions of DSPA and DPA, the challenge values $m$ and $d$ are assumed sampled at random (unrestrictedly to uniform distributions). One may consider "subset" versions of (second) preimage backdoors, i.e. where the backdoor only helps if the challenge value belongs to a specific subset; for example, one may design a hash for which only preimages of short strings—as passwords—can be found by the exploit algorithm.

Note that the notion of preimage dynamic adversaries are much closer to the original definitions of preimage resistance than the static versions. Roughly speaking, a dynamic preimage backdoor adversary exists only if trapdoor one-way functions (not necessarily permutations) exist, for the former is essentially a stealth version of the latter.

Again, the general definition of a distinguishing backdoor captures the previous definitions:

**Definition 7** (DDA)**.** *A* dynamic distinguishing adversary *for a relation* $\mathcal{R}$ *is a pair* (GenDD, ExpDD) *such that*

- *The* malicious generator $\mathsf{GenDD}$ *is a probabilistic algorithm that returns a pair* $(H, b)$*, where* $H$ *is a hash function and* $b$ *is a backdoor.*
- *The* exploit algorithm $\mathsf{ExpDD}$ *is a probabilistic algorithm that takes a hash function* $H$ *and a backdoor* $b$ *and that returns a list* $m_1, \dots, m_N$ *such that*

$$\mathcal{R}\left(m_1, \dots, m_N, H(m_1), \dots, H(m_N)\right) \ .$$

As in the definitions of dynamic (second) preimage adversaries, $\mathsf{ExpDD}$ can be seen as a sampler for the set of inputs satisfying the relation. Note that, depending on the relation $\mathcal{R}$ considered, the number of satisfying inputs may not be exponential. As with static distinguishers, relations should be non-trivial and convincing.

Our last definition is that of a key recovery backdoor, for some keyed hash function (e.g. HMAC):

**Definition 8** **(KRA).** *A* dynamic key recovery adversary *is a pair* $(\mathsf{GenKR}, \mathsf{ExpKR})$ *such that*

- *The* malicious generator $\mathsf{GenKR}$ *is a probabilistic algorithm that returns a pair* $(H, b)$*, where* $H$ *is a hash function and* $b$ *is a backdoor.*
- *The* exploit algorithm $\mathsf{ExpKR}$ *is a probabilistic algorithm that takes a hash function* $H$ *and a backdoor* $b$ *and that has oracle-access to* $H_K(\cdot)$ *for some key* $K$ *and that returns* $K$*.*

The definition assumed $K$ to be secret, and may be relaxed to subsets of "weak keys". This definition may also be relaxed to model forgery backdoors, i.e. adversaries that can forge MAC's (existentially, selectively, or universally) without recovering $K$.

## 2.3 Stealth definitions

We formalize the intuitive notions of undetectability ("Is there a backdoor?") and of undiscoverability ("What is the backdoor?") based on our definitions of static and dynamic adversaries. It is tempting to define undetectability in terms of indistinguishability between a malicious algorithm and a legit one. However, such a definition does not lend itself to a practical evaluation of hash algorithms. We thus relax the notion to define undetectablity as the inability to determine the exploit algorithm. In other words, it should be difficult to reverse-engineer the backdoor. We thus obtain the following definitions:

**Definition 9.** *A static collision backdoor* $(\mathsf{GenSC}, \mathsf{ExpSC})$ *is* undetectable *if given a* $H$ *returned by* $\mathsf{GenSC}$ *it is difficult to find* $\mathsf{ExpSC}$*.*

**Definition 10.** *A static preimage backdoor* $(\mathsf{GenSP}, \mathsf{ExpSP})$ *is* undetectable *if given a* $H$ *returned by* $\mathsf{GenSP}$ *it is difficult to find* $\mathsf{ExpSP}$*.*

And so on with respect to the other type of malicious hashes. For the sake of simplicity we do not specify the distribution of $H$'s. One may imagine stronger definitions wherein a number of distinct $H$'s and/or $\mathsf{GenSC}$ are available to the attackers. Subtleties may lie in the specification of $H$: one can imagine a canonical-form description that directly reveals the presence of the backdoor, while another description or implementation would make detection much more difficult. This issue is directly related to the notion of program obfuscation—be it practice- or theory-oriented—and to the impossibility of universal obfuscation [23]. In practice, however, algorithms are generally described in a simplified algorithmic form, which bypasses the problem of obfuscated programs. That said, we do not exclude the application of techniques from white-box cryptography or from software obfuscation (binary packers, etc.) to enhance undetectability or undiscoverability of a malicious hash.

Undiscoverability is more easily defined: it is the inability to find the backdoor $b$ when given the exploit algorithm. Example definitions are:

**Definition 11.** *A static distinguishing backdoor is* undiscoverable *if given* $\mathsf{ExpSD}$ *and a* $H$ *it is difficult to find the* $b$ *associated to* $H$*.*

**Definition 12.** *A dynamic collision backdoor is* undiscoverable *if given* $\mathsf{ExpDC}$ *and a* $H$ *it is difficult to find the* $b$ *associated to* $H$*.*

## 2.4 Remarks

**Relations between static and dynamic adversaries.** First, note that the notions of static collision and static preimage backdoors, as defined in §§2.1, are unrelated, i.e. none implies the other. Both allow the construction of a static distinguisher ($\mathsf{SDA}$).

Relations between notions of dynamic backdoors are similar to relations between classical security notions for hash functions, and share the same subtleties. Therefore, it is *in general* possible to construct $\mathsf{DSPA}$ from a $\mathsf{DPA}$ by setting $\mathsf{GenDSP} := \mathsf{GenDP}$ and

$$\mathsf{ExpDSP}(H, b, m) := \mathsf{ExpDP}(H, b, H(m)) \ .$$

In the same manner, a $\mathsf{DCA}$ is constructed from a $\mathsf{DSPA}$ by setting $\mathsf{GenDC} := \mathsf{GenDSP}$ and

$$\mathsf{ExpDC}(H, b) := \left[ m \overset{\$}{\leftarrow} \{0,1\}^n, \textbf{return } (m, \mathsf{ExpDSP}(H, b, m)) \right] \ .$$

As our definitions of dynamic backdoors are very general and flexible, we stress that the above reductions are not always possible (for example when the $\mathsf{DPA}$ only works for a subset of the digests). That said, rigorous definitions *à la* Rogaway [24] are unnecessary for our purposes.

**Backdoors for compression functions.** We only discussed backdoors for a hash function rather than for its building blocks. Below we briefly discuss generalizations to compression functions as well as relations with backdoors on the hash function constructed.

For simplicity, and to model real use-cases, we consider a compression function as a function mapping a message chunk to an internal state, i.e. implicitly assuming a fixed IV. This allows us to capture both Merkle-Damgård (as well as HAIFA, wide-pipe, et al.) and sponge functions. Definitions in §2 directly extend to such a compression function. Examples of relations between backdoors for compression functions and for the hash are

- An $\mathsf{SCA}$ for the compression function is sufficient to construct a $\mathsf{DCA}$ for the full hash: an internal collision can indeed be exploited to produce an arbitrary number of colliding messages, as in the "poisoned message attack". Note that this is inapplicable to the rare constructions that incorporate a "checksum" (as the GOST hash function).
- A $\mathsf{DSPA}$ for the compression function can be used to construct a $\mathsf{DSPA}$ for the full hash. However, a $\mathsf{DPA}$ for the compression function does not necessarily transfers to the hash, e.g. in constructions that incorporate a finalization function and in sponge functions with $n > b$ (i.e. with a multi-block output).

As we will see below, a backdoor can be planted either in the iterated compression function, in the finalization function, or in the mode of operation and control logic of the implementation.

## 3 Strategies

There are two main approaches to create malicious hash functions:

- *A priori design*: create a new design from scratch with as target classical security notions as well as one or more malicious notions.
- *A posteriori design*: incorporate a backdoor in an existing design, via a "tweak", an "improved version", or a malicious implementation. For instance, an "improved" message expansion for SHA256 can easily dissimulate a backdoor, as well as a new set of word constants.

Below we overview more specific strategies, though without providing details.

### 3.1 Black-box backdoors

Young and Yung investigaged [19] malicious block ciphers in the black-box model, i.e. where the description of the cipher is unknown to the adversary. In this weak attack model, simple and powerful backdoors can be implemented in hash functions (or in implementations thereof), in the same spirit as bug attacks [10]. For example, let's construct a DPA for the future SHA3:

- GenDP defines the backdoor as an arbitrary string $b$ of (say) 128 bits, and the malicious function bSHA3-256 as follows[5]:
  - Given an $N$-block padded message $m$, process it with SHA3 up to its penultimate block $m_{N-2}$;
  - Process the last block $m_{N-1}$ as per SHA3-256; if the first 128 bits of $m_{N-1}$ are equal to $b$, then return the 256 bits following $b$, otherwise return the digest computed.
- ExpDP, given $b$, and a target digest $d$, chooses an arbitrary message of length a multiple of the block size, appends $b$ and $d$ to it, and verifies that bSHA3-256 returns $d$ as a digest.

Such a backdoor can be used to find preimages representing valid file formats as PDF or JPEG by placing the malicious data after the end-of-file symbol. Another advantage is that the function appears as a legitimate version of (say) SHA256, until one submits one of the backdoor strings. Disadvantages are that $b$ and $d$ appear as substrings of the last block, which may raise alarms; to avoid this problem, $b$ and $d$ may be encoded (e.g. as noisy codewords of some appropriate error-correcting code) or encrypted. Finally, implementations of such malicious hash functions should be careful to avoid trivial side-channel attacks, such as timing attacks (note that our ExpDP makes a dummy call to the compression function to keep time-constant computations).

### 3.2 General strategies

General strategies for non-blackbox backdoors consist in *which* component or building block includes the backdoor, and *what* is the general target of the attacker, rather than *how*, which is the subject of the next subsection. Below we discuss some examples of general strategies:

[5]The b-prefix notation is inspired from Rijmen-Preneel [13].

- *Backdoor-in-the-middle*: this strategy modifies the internals of a permutation so as to connect given inputs to a given outputs. It applies as well to block-cipher based hashes as to sponge functions.
- *Malicious finalization*: this strategy consists in designing an output filter, typically independent from the hashed data, which behaves in a malicious way for some precomputed inputs that come from a legitimate function.
- *Weak mode trigger*: this strategy consists in designing a hash that enters a "weak mode" (typically, a weak internal state) upon some predetermined input. Eve's attack then consists in 1) exploiting the backdoor to enter a weak state and 2) exploiting the said weakness.

The first two strategies are directly applicable to static collision and preimage backdoors, while the third is more oriented toward dynamic attacks. Note that the trivial approach of using a trapdoor one-way permutation (e.g. within a sponge) to realize dynamic preimage backdoors leads to easily detectable backdoors, unless (say) an ARX trapdoor function is discovered.

Examples of recent hash functions that lend themselves to a posteriori malicious finalizations are the SHA3 candidates SIMD and Grøstl, which use a message-independent output filter. Observe the the entropy reduction (i.e. truncation) leaves room for collision backdoors.

A simple strategy of the third type, for a Merkle-Damgård hash, consists in exploiting a precomputed fixed point $E_m(h) \oplus h = h$ as IV, in order to compute second preimages. The SHA3 candidate CubeHash is an example of function with "weak states", that may be exploited in the creation of a backdoored version.

Strategies for compression-based constructions significantly differ from strategies for sponge functions: for example, input-specific behavior is difficult to implement in typical compression functions, for message bits are injected regularly to the internal state; this is not the case for sponges, which seems to facilitate the design of some backdoors. Moreover, collisions (and preimages) for sponges do not require that all the internal state match the expected value, but only $c - r$ bits (where $c$ is the capacity and $r$ the bit rate of the sponge).

## 3.3 Strategies to achieve stealth

Malicious hash functions will typically require a less symmetric and structured construction, due to the entropy necessary to generate the hash instance. There are basically two approaches to maximize undetectability, for which we propose dedicated terms:

- *Entropy spraying*, or "needles in a haystack", i.e. bits of entropy are dispersed within a well-organized and clean structure, that should be as large as possible.
- *Chameleon structure*, or "needles in a needlestack", i.e. entropy is part of an algorithm that lacks any structure or consistency, similarly to the Hasty Pudding Cipher.

## 3.4 Specific strategies: neutral structures

A basic strategy to trigger a specific behavior of a function is to choose operations such that—to simplify—a given output maps to a given output (for preimages), or that two given inputs map to a same output (for collisions). We introduce the term *neutral structure* to denote a tree whose nodes are undetermined operators, leaves are a subset of the internal state, and the root node is the output. Freedom degrees lie in the choice of the operators. A simple example of neutral structure for a permutation with a 4-word internal state $s = (s_0, s_1, s_2, s_3)$ is

$$\begin{aligned} s_0 &= s_1 + x \\ s_1 &= s_2 \oplus y \\ s_2 &= s_3 + z \\ s_3 &= s_0 \star ((s_1 \bullet s_2) \diamond (s_3 \ggg n) \end{aligned}$$

where $x$, $y$, $z$ are values to be taken from a set of $C$ unsuspicious constants (pi's digits, `DEADBEEF`'s, etc.), and binary operators $\star$, $\bullet$ and $\diamond$ are to be chosen among $B$ common operations and $n$ is a rotation distance in $\{1, \dots, 31\}$. This neutral structure thus has entropy approximately

$$3 \log_2 C + 3 \log_2 B + \log_2(31) \ .$$

The use of word constants as freedom degrees is in general straighforward, however it makes undetectability more difficult.

Neutral structures determined by bruteforce should have an output short enough so that search is efficient. For example, a neutral structure composed of only ARX operations could return a single word, as a function of (part of) the internal state. Neutral structures can be based on S-boxes, arithmetic operations, or any operations used in cryptographic algorithms.

## 4 Example: maliciousеd BLAKE

We briefly describe how to plant a simple dynamic collision backdoor in the SHA3 finalist BLAKE, whose compression function lends itself well to malicious finalization, due to its "local wide-pipe" construction.

### 4.1 Brief description of BLAKE-256's compression function

The compression function of BLAKE-256 initializes an internal state of sixteen 32-bit words from an 8-bit chaining value, a salt, and a counter. This 512-bit state is transformed by a permutation and the new 256-bit chaining value $h_0, \dots, h_7$ is updated as as $h_i += v_i \oplus v_{i+8}$, where $v_i$'s are internal state words. We refer to the official documentation of BLAKE for a complete specification.

### 4.2 Neutral structure for static collisions

It is easy to see that if BLAKE's permutation is secure, then such a simple finalization transform is sufficient. However, for some reasons, Eve may claim to use a more "secure" (but in fact only more complicated) finalization in the compression function. Eve may thus use the following neutral structure to obtain a compression function for which the two one-block messages of her choice collide:

$$S_i(v) = (v_i \bullet v_{i+1}) \ggg r_{i,1} \bullet (v_{i+2} \bullet v_{i+3}) \ggg r_{i,2} \bullet (v_{i+4} \bullet v_{i+5}) \ggg r_{i,3} \bullet (v_{i+6} \bullet v_{i+7}) \ggg r_{i,4} \bullet (v_{i+8} \bullet v_{i+9}) \ggg r_{i,5}$$

where indices are to be reduced modulo 16. We have 8 binary operators either "+" or "$\oplus$", and 5 rotations with each 31 possibles distances; hence in total approximately $2^8 \times 32^5 = 2^{33}$ choices. The entropy of this neutral structure is thus sufficient to find eight instances of the structure $S_0, \dots, S_7$ that safisfy

$$S_i(v) = S_i(v'), \quad i = 0, \dots, 7 \ ,$$

for two fixed internal states $v$ and $v'$ derived from chosen messages. It is clear that if BLAKE is secure, then this variant is secure as well; informally, finding the backdoor collision is much harder than finding a simple collision; and would one know the colliding final states, one would have to find the two preimages that lead to this state.

To find a BLAKE-256 collision between the two strings “YES” and “NO” (null-terminated), we found the following finalization function:

$$\begin{gathered}
((v_0 \oplus v_1) \ggg 11) + ((v_2 \oplus v_3) \ggg 18) + ((v_4 \oplus v_5) \ggg 11) \oplus ((v_6 + v_7) \ggg 20) \oplus ((v_8 + v_9) \ggg 19) \\
((v_1 \oplus v_2) \ggg 17) + ((v_3 \oplus v_4) \ggg 16) + ((v_5 \oplus v_6) \ggg 28) \oplus ((v_7 + v_8) \ggg 10) + ((v_9 + v_{10}) \ggg 30) \\
((v_2 \oplus v_3) \ggg 12) + ((v_4 + v_5) \ggg 17) \oplus ((v_6 \oplus v_7) \ggg 13) \oplus ((v_8 + v_9) \ggg 22) \oplus ((v_{10} \oplus v_{11}) \ggg 7) \\
((v_3 \oplus v_4) \ggg 7) \oplus ((v_5 \oplus v_6) \ggg 5) \oplus ((v_7 + v_8) \ggg 11) + ((v_9 + v_{10}) \ggg 2) \oplus ((v_{11} + v_{12}) \ggg 9) \\
((v_4 \oplus v_5) \ggg 6) + ((v_6 + v_7) \ggg 6) + ((v_8 \oplus v_9) \ggg 4) + ((v_{10} \oplus v_{11}) \ggg 21) \oplus ((v_{12} + v_{13}) \ggg 15) \\
((v_5 + v_6) \ggg 4) + ((v_7 + v_8) \ggg 30) + ((v_9 \oplus v_{10}) \ggg 30) + ((v_{11} + v_{12}) \ggg 29) + ((v_{13} \oplus v_{14}) \ggg 2) \\
((v_6 \oplus v_7) \ggg 22) \oplus ((v_8 \oplus v_9) \ggg 1) \oplus ((v_{10} + v_{11}) \ggg 30) \oplus ((v_{12} \oplus v_{13}) \ggg 22) + ((v_{14} + v_{15}) \ggg 21) \\
((v_7 + v_8) \ggg 19) \oplus ((v_9 + v_{10}) \ggg 8) + ((v_{11} + v_{12}) \ggg 25) \oplus ((v_{13} \oplus v_{14}) \ggg 15) \oplus ((v_{15} \oplus v_0) \ggg 10)
\end{gathered}$$

## Acknowledgments

We thank the reviewers of the ECRYPT2 Hash 2011 workshop for their insightful comments, and Gaëtan Leurent for the fixed-point trick and other ideas.

## References

1. Andrews, J.: Linux: Kernel ”back door” attempt. KernelTrap (2003) `http://kerneltrap.org/node/1584`.
2. Shumow, D., Ferguson, N.: On the possibility of a back door in the NIST SP800-90 Dual Ec Prng. CRYPTO 2007 rump session (2007)
3. NIST: Recommendation for random number generation using deterministic random bit generators (revised). NIST Special Publication 800-90 (2007)
4. Schoenmakers, B., Sidorenko, A.: Cryptanalysis of the dual elliptic curve pseudorandom generator. Cryptology ePrint Archive, Report 2006/190 (2006)
5. Brown, D.R.L., Gjøsteen, K.: A security analysis of the NIST SP 800-90 elliptic curve random number generator. Cryptology ePrint Archive, Report 2007/048 (2007)
6. Bamford, J.: The Puzzle Place. Penguin (1983)
7. Schneier, B.: Applied Cryptography. 2nd edn. Wiley (1996)
8. Braeckeleer, L.D.: For years US eavesdroppers could read encrypted messages without the least difficulty. The Intelligence Daily (Dec. 2007) `http://www.inteldaily.com/?c=169&a=4686`.
9. Wagner, D., Bionbi, P.: Misimplementation of RC4. Submission for the Third Underhanded C Contest (2007) `http://underhanded.xcott.com/?page_id=16`.
10. Biham, E., Carmeli, Y., Shamir, A.: Bug attacks. In: CRYPTO. (2008)
11. Young, A., Yung, M.: Malicious Cryptography: Exposing Cryptovirology. Wiley (2004)
12. Bishop, M., Engle, S., Peisert, S., Whalen, S., Gates, C.: We have met the enemy and he is us. In: New Security Paradigms, ACM (2008) 1–12
13. Rijmen, V., Preneel, B.: A family of trapdoor ciphers. In: FSE. (1997)
14. Wu, H., Bao, F., Deng, R.H., Ye, Q.Z.: Cryptanalysis of Rijmen-Preneel trapdoor ciphers. In: ASIACRYPT. (1998)
15. Blaze, M., Feigenbaum, J., Leighton, T.: Master key cryptosystems. CRYPTO 1995 rump session (1995)
16. Patarin, J., Goubin, L.: Trapdoor one-way permutations and multivariate polynominals. In: ICICS. (1997)
17. Ye, D., Lam, K.Y., Dai, Z.D.: Cryptanalysis of “2 R” schemes. In: CRYPTO. (1999)
18. Biham, E.: Cryptanalysis of Patarin’s 2-round public key system with S boxes (2R). In: EUROCRYPT. (2000)
19. Young, A., Yung, M.: Monkey: Black-box symmetric ciphers designed for monopolizing keys. In: FSE. (1998)

20. Young, A., Yung, M.: Backdoor attacks on black-box ciphers exploiting low-entropy plaintexts. In: ACISP. (2003)
21. Filiol, E.: Malicious cryptography techniques for unreversable (malicious or not) binaries. CoRR **abs/1009.4000** (2010)
22. Wysopal, C., Eng, C.: Static detection of application backdoors. `http://www.veracode.com/images/stories/static-detection-of-backdoors-1.0.pdf` Presented at BlackHat USA 2007.
23. Barak, B., Goldreich, O., Impagliazzo, R., Rudich, S., Sahai, A., Vadhan, S.P., Yang, K.: On the (im)possibility of obfuscating programs. In: CRYPTO. (2001)
24. Rogaway, P.: Nonce-based symmetric encryption. In: FSE. (2004)