Malicious Hashing: Eve's Variant of SHA-1

Ange Albertini[1], Jean-Philippe Aumasson[2], Maria Eichlseder[3](✉), Florian Mendel[3], and Martin Schläffer[3]

[1] Corkami, Ravensburg, Germany
ange.albertini@gmail.com
[2] Kudelski Security, Cheseaux-sur-Lausanne, Switzerland
jeanphilippe.aumasson@gmail.com
[3] Graz University of Technology, Graz, Austria
{maria.eichlseder,florian.mendel,martin.schlaeffer}@iaik.tugraz.at

Abstract. We present collisions for a version of SHA-1 with modified constants, where the colliding payloads are valid binary files. Examples are given of colliding executables, archives, and images. Our malicious SHA-1 instances have round constants that differ from the original ones in only 40 bits (on average). Modified versions of cryptographic standards are typically used on closed systems (e.g., in pay-TV, media and gaming platforms) and aim to differentiate cryptographic components across customers or services. Our proof-of-concept thus demonstrates the exploitability of custom SHA-1 versions for malicious purposes, such as the injection of user surveillance features. To encourage further research on such malicious hash functions, we propose definitions of malicious hash functions and of associated security notions.

1 Introduction

In 2013, cryptography made the headlines following the revelation that NSA may not only have compromised cryptographic software and hardware, but also cryptographic algorithms. The most concrete example is the "key escrow" [11] or "master key" [5] property of the NSA-designed `Dual_EC_DRBG` [27]. The alleged backdoor is the number e such that $eQ = P$, where P and Q are two points on the elliptic curve specified as constants in `Dual_EC_DRBG`. Knowing e allows one to determine the internal state and thus to predict all future outputs. Despite other issues [7,35] (see also [16,17]), `Dual_EC_DBRG` was used as default pseudorandom number generator in EMC/RSA's BSAFE library, allegedly following a $10M deal with NSA [24].

It is also speculated that NSA may have "manipulated constants" [34] of other algorithms, although no hard evidence has been published. This series of revelations prompted suspicions that NIST-standardized cryptography may be compromised by NSA. It also raised serious doubts on the security of commercial cryptography software, and even of open-source software. Several projects have been started to address those concerns, like #youbroketheinternet [43] and the Open Crypto Audit Project [29].

A. Joux and A. Youssef (Eds.): SAC 2014, LNCS 8781, pp. 1–19, 2014.
DOI: 10.1007/978-3-319-13051-4_1

Research on cryptographic backdoors and malicious cryptography appears to have been the monopoly of intelligence agencies and of industry. Only a handful of peer-reviewed articles have been published in the open literature (see Sect. 2.1), whereas research on related topics like covert-channel communication (e.g., [26,36]) or hardware trojans (e.g., [2,20]) is published regularly.

Malicious ciphers have been investigated by Young and Yung [44] in their "cryptovirology" project, and to a lesser extent by Rijmen and Preneel [33] and Patarin and Goubin [30]. However we are unaware of any research about *malicious hash functions*—that is, hash functions designed such that the designer knows a property that allows her to compromise one or more security notions. Note that we distinguish backdoors (covert) from trapdoors (overt); for example VSH [9] is a trapdoor hash function, such that collisions can be found efficiently if the factorization of the RSA modulus is known.

This paper thus investigates malicious hash functions: first their definition and potential applications, then a proof-of-concept by constructing a malicious version of SHA-1 with modified constants, for which two binary files (e.g., executables) collide. We have chosen SHA-1 as a target because it is (allegedly) the most deployed hash function and because of its background as an NSA/NIST design. We exploit the freedom of the four 32-bit round constants of SHA-1 to efficiently construct 1-block collisions such that two valid executables collide for this malicious SHA-1. Such a backdoor could be trivially added if a new constant is used in every step of the hash function. However, in SHA-1 only four different 32-bit round constants are used within its 80 steps, which significantly reduces the freedom of adding a backdoor. Actually our attack only modifies at most 80 (or, on average, 40) of the 128 bits of the constants.

Our malicious SHA-1 can readily be exploited in applications that use custom hash functions (e.g., for customers' segmentation) to ensure that a legitimate application can surreptitiously be replaced with a malicious one, while still passing the integrity checks such as secure boot or application code signing.

Outline. Section 2 attempts to formalize intuitive notions of malicious hash functions. We first define a malicious hash function as a pair of algorithms: a *malicious generator* (creating the function and its backdoor) and an *exploit algorithm*. Section 3 then presents a novel type of collision attack, exploiting the freedom degrees of the SHA-1 constants to efficiently construct collisions. We describe the selection of a dedicated disturbance vector that minimizes the complexity and show examples of collisions. Section 4 discusses the application to structured file formats, and whether the constraints imposed by the attack can be satisfied with common file formats. We present examples of valid binary files that collide for our malicious SHA-1: executables (e.g., master boot records or shell scripts), archives (e.g., rar), and images (jpg).

2 Malicious Hashing

We start with an overview of previous work related to malicious cryptography. Then we formalize intuitive notions of malicious hashing, first with a general definition of a malicious hash function, and then with specific security notions.

2.1 Malicious Cryptography and Backdoors

The open cryptographic literature includes only a handful of works related to malicious applications of cryptography:

- In 1997, Rijmen and Preneel [33] proposed to hide linear relations in S-boxes and presented "backdoor versions" of CAST and LOKI. These were broken in [41] as well as the general strategy proposed. Rijmen and Preneel 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." [33]. Indeed, [5] previously argued that a backdoor block cipher is equivalent to a public key cryptosystem.
- In 1997, Patarin and Goubin [30] proposed an S-box-based asymmetric scheme constructed as a 2-round SPN but publicly represented as the corresponding equations – keeping the S-boxes and linear transforms secret. This was broken independently by Ye et al. and Biham [3,42]. This can be seen as an ancestor of white-box encryption schemes.
- In 1998 and later, 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 through a covert channel [45,46]. Young and Yung coined the term *cryptovirology* [44] and cited various malicious applications of cryptography: ransomware, deniable data stealing, etc.
- In 2010, Filiol [15] proposed to use malicious pseudorandom generators to assist in the creation of executable code difficult to reverse-engineer. Typical applications are the design of malware that resist detection methods that search for what looks like obfuscated code (suggesting the hiding of malicious instructions).

Note that we are concerned with backdoors in *algorithms*, regardless of its representation (pseudocode, assembly, circuit, etc.), as opposed to backdoors in software implementations (like Wagner and Biondi's sabotaged RC4 [38]) or in hardware implementations (like bug attacks [4] and other hardware trojans).

2.2 Definitions

We propose 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.

Malicious Hash Function. Contrary to typical security definitions, our adversary is not the attacker, so to speak: instead, the adversary Eve *creates* the primitive and knows the secret (i.e. the backdoor and how to exploit it), whereas honest parties (victims) attempt to cryptanalyze Eve's design. We thus define a malicious hash function (or adversary) as a pair of efficient algorithms, modeling the ability to create malicious primitives and to exploit them:

- 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 distinguish two types of backdoors: *static*, which have a deterministic exploit algorithm, and *dynamic*, which have a probabilistic one.

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

Static Backdoors Adversaries. Eve is a *static collision adversary* (SCA) if she designs a hash function for which she knows one pair of colliding messages.

Definition 1 **(SCA).** *A* static collision adversary *is a pair* (GenSC, ExpSC) *such that*

- *The* malicious generator 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 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 an adversary producing a small number of collisions, through the definition of several ExpSC algorithms $\mathsf{ExpSC}_1, \ldots, \mathsf{ExpSC}_n$.

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

Definition 2 **(SPA).** *A* static preimage adversary *is a pair* (GenSP, ExpSP) *such that*

- *The* malicious generator 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 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 informally defined as digest having a pattern that will convince a third party that "something is wrong" with the hash function; for example, the all-zero digest, a digest with all bytes identical, etc.

Dynamic Backdoors. Dynamic backdoors 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 MACs.

Definition 3 (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, ExpDC 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 dynamic second-preimage and preimage adversaries follow naturally:

Definition 4 (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 5 (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 (unrestricted 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.

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

Definition 6 (KRA). *A* dynamic key-recovery adversary *is a pair* (GenKR, ExpKR) *such that*

- *The* malicious generator 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 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 of KRA assumes 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.

Stealth Definitions. We attempt to formalize the intuitive notions of undetectability ("Is there a backdoor?") and of undiscoverability ("What is the backdoor?"). 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 (that is, how the backdoor works, regardless of whether one knows the necessary information, b). In other words, it should be difficult to reverse-engineer the backdoor. We thus have the following definition, applying to both collision and preimage backdoors:

Definition 7. *The backdoor in a malicious hash* (Gen, Exp) *is* undetectable *if given a H returned by* Gen *it is difficult to find* Exp.

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 obfuscation (be it at the level of the algorithm, source code, intermediate representation, etc.). For example, malware (such as ransomware, or jailbreak kits) may use obfuscation to dissimulate malicious features, such as cryptographic components of 0-day exploits.

Furthermore, backdoors may be introduced as sabotaged versions of legitimate designs. In that case, undetectability can take another sense, namely distinguishability from the original design. For example, in our malicious SHA-1, it is obvious that the function differs from the original SHA-1, and one may naturally suspect the existence of "poisonous" inputs, although those should be hard to determine.

Undiscoverability is more easily defined than undetectability: it is the inability to find the backdoor b given the exploit algorithm. A general definition is as follows:

Definition 8. *The backdoor in a malicious hash* (Gen, Exp) *is* undiscoverable *if given* Exp *and H returned by* Gen *it is difficult to find b.*

In our proof-of-concept of a malicious SHA-1, undiscoverability is the hardness to recover the colliding pair, given the knowledge that a pair collides (and even the differential used).

3 Eve's Variant of SHA-1

As a demonstration of the above concepts, we present an example of a *static collision backdoor*: Eve constructs a custom variant of SHA-1 that differs from the standardized specification only in the values of some round constants (up to 80 bits). Eve can use the additional freedom gained from choosing only four 32-bit constants to find a practical collision for the full modified SHA-1 function during its design. We show that Eve even has enough freedom to construct a meaningful collision block pair which she can, at a later point, use to build multiple colliding file pairs of a particular format (e.g., executable or archive format) with almost arbitrary content.

The backdoor does not exploit any particular "weaknesses" of specific round constants, nor does it weaken the logical structure of the hash function. Instead, it only relies on the designer's freedom to choose the constants during the attack. This freedom can be used to improve the complexity of previous attacks [37,40] and thus makes it feasible to find collisions for the full hash function.

For an attacker who only knows the modified constants but cannot choose them, collisions are as hard to find as for the original SHA-1. Thus, in terms of the definitions of the previous section, this backdoor is *undiscoverable*. It is, however, *detectable* since constants in hash functions are normally expected to be identifiable as nothing-up-your-sleeve numbers. This is hardly achievable in our attack.

Below, we first give a short description of SHA-1 in Sect. 3.1 and briefly review previous differential collision attacks on SHA-1 in Sect. 3.2. Then, we build upon these previous differential attacks and describe how the freedom of choosing constants can be used to improve the attack complexity in Sect. 3.3.

3.1 Short Description of SHA-1

SHA-1 is a hash function designed by the NSA and standardized by NIST in 1995. It is an iterative hash function based on the Merkle-Damgård design principle [10,25], processes 512-bit message blocks and produces a 160-bit hash value by iterating a compression function f. For a detailed description of SHA-1 we refer to [28].

The compression function f uses the Davies-Meyer construction which consists of two main parts: the message expansion and the state update transformation. The message expansion of SHA-1 is a linear expansion of the 16 message words (denoted by M_i) to 80 expanded message words W_i,

$$W_i = \begin{cases} M_i & \text{for } \; 0 \leq i \leq 15, \\ (W_{i-3} \oplus W_{i-8} \oplus W_{i-14} \oplus W_{i-16}) \lll 1 & \text{for } 16 \leq i \leq 79\ . \end{cases}$$

The state update transformation of SHA-1 consists of 4 rounds of 20 steps each. In each step, the expanded message word W_i is used to update the 5 chaining variables as depicted in Fig. 1. In each round, the step update uses different Boolean functions f_r and additive constants K_r, which are shown in Table 1.

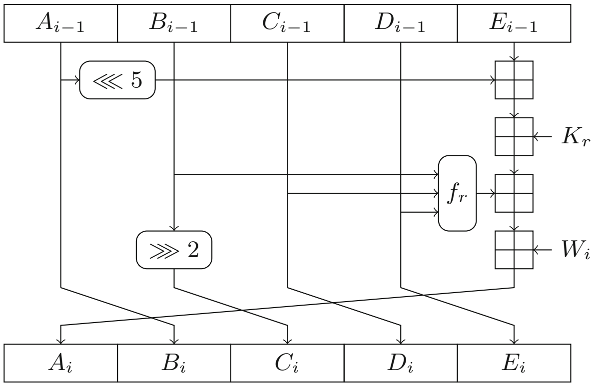

Fig. 1. The step function of SHA-1.

Table 1. The round constants K_r and Boolean functions f_r used in each step of SHA-1.

Round r	Step i	K_r	f_r
1	$0 \leq i \leq 19$	`5a827999`	$f_{\mathrm{IF}}(B, C, D) = B \wedge C \oplus \neg B \wedge D$
2	$20 \leq i \leq 39$	`6ed9eba1`	$f_{\mathrm{XOR}}(B, C, D) = B \oplus C \oplus D$
3	$40 \leq i \leq 59$	`8f1bbcdc`	$f_{\mathrm{MAJ}}(B, C, D) = B \wedge C \oplus B \wedge D \oplus C \wedge D$
4	$60 \leq i \leq 79$	`ca62c1d6`	$f_{\mathrm{XOR}}(B, C, D) = B \oplus C \oplus D$

For Eve's modified, malicious hash function, we only change the values of K_2, K_3 and K_4. The remaining definition is left unchanged. Note that the original SHA-1 constants are chosen as the square roots of 2, 3, 5 and 10:

$$K_1 = \lfloor \sqrt{2} \cdot 2^{30} \rfloor, \quad K_2 = \lfloor \sqrt{3} \cdot 2^{30} \rfloor, \quad K_3 = \lfloor \sqrt{5} \cdot 2^{30} \rfloor, \quad K_4 = \lfloor \sqrt{10} \cdot 2^{30} \rfloor.$$

3.2 Differential Attack Strategy for SHA-1

At CRYPTO 2005, Wang et al. presented the first collision attack on full SHA-1 with a theoretical complexity of about 2^{69} [40]. This was later improved to 2^{63} by the same authors [39]. Since then, several papers on the cryptanalysis of SHA-1 have been published [1,13,14,19,37]. Nevertheless, no practical collision has been shown for full SHA-1 to date.

Our practical and meaningful collision attacks on malicious SHA-1 are based on the differential attacks by Wang et al. in [40] and its improvements. In a differential collision attack, we first construct a high-probability differential characteristic that yields a zero output difference, i.e., a collision. In the second stage, we probabilistically try to find a confirming message pair for this differential characteristic.

By using a differential characteristic with a lower probability at the beginning of the hash function (first round of SHA-1), the probability of the remaining characteristic can be further improved. Since the message can be chosen freely in a hash function attack, we can significantly improve the complexity of finding confirming message pairs at the beginning of the hash function using message modification techniques [40]. A high-level overview of such a differential attack on SHA-1 is given as follows:

1. **Find a differential characteristic**
   (a) Construct the high-probability part
   (b) Determine the low-probability part
2. **Find a confirming message pair**
   (a) Use message modification in low-probability part
   (b) Perform random trials in high-probability part

The high-probability part of the differential characteristic for SHA-1 covers round 2 to round 4. It has been shown in [8,21,40] that for SHA-1, the best way to construct these high-probability characteristics is to interleave so-called local collisions (one disturbing and a set of correcting differences). These characteristics can be easily constructed by using a linearized variant of the hash function and tools from coding theory [31,32]. The probability of this characteristic determines the complexity of the attack on SHA-1.

The low-probability part and message modification take place in round 1 and are typically performed using automated non-linear equation solving tools [14,22,23]. We stress that the total complexity is still above 2^{60} for all published collision attacks so far, which is only practical for attackers with large computing power (NSA, Google, etc.).

3.3 Malicious Collision Attack

In SHA-1, a new 32-bit constant $K_1, \ldots, K_4$ is used in each of the four rounds. In our malicious collision attack, we use the freedom of these four constants to reduce the complexity of the attack. Similar to message modification, we choose the constants during the search for a confirming message pair. We modify the constants in a round-by-round strategy, always selecting a round constant such that the differential characteristic for the steps of the current round can be satisfied. Since we have to choose the constants when processing the first block, we can only improve the complexity of this block. Hence, we need to use a differential characteristic that results in a single-block collision. Note that all the collisions attacks on SHA-1 so far use a 2-block characteristic.

To find the high-probability part of a differential characteristic for round 2–4 resulting in a 1-block collision, a linearized variant of the hash function can be used. However, using algorithms from coding theory, we only find differential characteristics that maximize the overall probability and do not take the additional freedom we have in the choice of the constants in SHA-1 into account. Therefore, to minimize the overall attack complexity, we did not use these differential characteristics. Instead, we are interested in a differential characteristic such that the minimum of the three probabilities for round 2, 3 and 4 is maximized. To find such a characteristic, we start with the best overall characteristic and modify it to suit our needs.

In previous attacks on SHA-1, the best differential characteristics for rounds 2–4 have differences only at bit position 2 for some 16 consecutive state words A_i [21]. We assume that the best differential characteristic has the same property in our case. Hence, we only need to determine all 2^{16} possible differential characteristics with differences only at bit position 2 in 16 consecutive state words A_i and linearly expand them backward and forward. A similar approach has also been used to attack SHA-0 [8] and SHA-1 [21,40].

For each of these 2^{16} differential characteristics, we estimate the cost of finding a malicious single-block collision. These costs are roughly determined by the number of differences (disturbances) in A_i in each round. For details on the cost computations, we refer to [31]. The estimated costs for the best differential characteristics suited for our attack are given in Table 2, and the correspond message differences are given in Table 3.

Table 2. Probabilities for rounds 2–4 of the differential characteristics suitable for our attack.

Candidate	$r = 2$	$r = 3$	$r = 4$	Total
MD_1	2^{-40}	2^{-40}	2^{-15}	2^{-95}
MD_2	2^{-39}	2^{-42}	2^{-13}	2^{-94}
MD_3	2^{-39}	2^{-42}	2^{-11}	2^{-92}

Table 3. List of message differences suitable for our attack

MD_1	00000003	20000074	88000000	e8000062	c8000043	28000004	40000042	48000046
	88000002	00000014	08000002	a0000054	88000002	80000000	a8000003	a8000060
MD_2	20000074	88000000	e8000062	c8000043	28000004	40000042	48000046	88000002
	00000014	08000002	a0000054	88000002	80000000	a8000003	a8000060	00000003
MD_3	88000000	e8000062	c8000043	28000004	40000042	48000046	88000002	00000014
	08000002	a0000054	88000002	80000000	a8000003	a8000060	00000003	c0000002

The high-probability differential characteristic with message difference MD_1 is best suitable for our intended file formats (see Sect. 4) and used as the starting point to search for a low-probability differential characteristic for the first round of SHA-1. We use an automatic search tool [22,23] to find the low-probability part of the characteristic. The result is shown in Table 5 in the appendix. Overall, the complexity of finding a colliding message pair using malicious constants for this differential characteristic in our attack is approximately 2^{48}, which is feasible in practice as demonstrated below and in Sect. 4.

After the differential characteristic is fixed, we probabilistically search for a confirming message pair. We start with only the first constant K_1 fixed (e.g., to the standard value) and search for a message pair that confirms at least the first round (20 steps) of the characteristic, and is also suitable for our file format. This is easier than finding a message pair that works for all four rounds (with fixed constants), since fewer constraints need to be satisfied. The complexity of this step is negligible.

Now, we can exhaustively search through all 2^{32} options for K_2 until we find one that confirms round 2. Only if no such constant is found, we backtrack and modify the message words. Since the differential characteristic for message difference MD_1 holds with probability 2^{-40} in round 2 and we can test 2^{32} options for K_2, this step of the attack will only succeed with a probability of 2^{-8}. Hence, completing this step alone has a complexity of approximately 2^{40}.

Once we have found a candidate for K_2 such that the differential characteristic holds in round 2, we proceed in the same way with K_3. Again, the differential characteristic will hold with only a probability of 2^{-40} in round 3 and we can test only 2^{32} options for K_3. Therefore, we need to repeat the previous steps of the attack 2^8 times to find a solution. Including the expected 2^8 tries for the previous step to reach the current one, completing this step has an expected complexity of roughly 2^{48}.

Finally, we need to find K_4. Since the last round of the characteristic has a high probability, such a constant is very likely to exist and this step of the attack only adds negligible cost to the final attack complexity of about 2^{48}.

Normally, with fixed constants, an attacker would have to backtrack in the case of a contradiction in the later steps. Eve as the designer, on the other hand, has a chance that choosing a different constant might repair the contradictions for another round. This significantly improves the complexity of the differential attack. For predefined constants, the complexity of the attack for this particular disturbance vector would be roughly 2^{95}.

Note that we do not need the whole freedom of all 4 constants. The first constant in round 1 can be chosen arbitrarily (e.g., we keep it as in the original SHA-1 specification). For the last constant in round 4, we can fix approximately 16 bits of the constant. That is, 80 bits of the constants need to be changed compared to the original values. More freedom in choosing the constants is possible if we increase the attack complexity. An example of a colliding message pair for our malicious SHA-1 variant with modified constants is given in Table 4. The constants differ from the original values by 45 (of 128) bits. In the following section, we will show how this pair can be used to construct meaningful collisions.

4 Building Meaningful Collisions

To exploit the malicious SHA-1 described in Sect. 3, we propose several types of executable, archive and image file formats for which two colliding files can be created, and such that the behavior of the two files can be fully controlled by the attacker.

Table 4. Example of a collision for SHA-1 with modified constants $K_{1...4}$.

$K_{1...4}$	5a827999 4eb9d7f7 bad18e2f d79e5877
IV	67452301 efcdab89 98badcfe 10325476 c3d2e1f0
m	ffd8ffe1 e2001250 b6cef608 34f4fe83 ffae884f afe56e6f fc50fae6 28c40f81
	1b1d3283 b48c11bc b1d4b511 a976cb20 a7a929f0 2327f9bb ecde01c0 7dc00852
m^*	ffd8ffe2 c2001224 3ecef608 dcf4fee1 37ae880c 87e56e6b bc50faa4 60c40fc7
	931d3281 b48c11a8 b9d4b513 0976cb74 2fa929f2 a327f9bb 44de01c3 d5c00832
Δm	00000003 20000074 88000000 e8000062 c8000043 28000004 40000042 48000046
	88000002 00000014 08000002 a0000054 88000002 80000000 a8000003 a8000060
$h(m)$	1896b202 394b0aae 54526cfa e72ec5f2 42b1837e

Below, we first discuss the constraints that the files have to satisfy, in order to collide with our malicious SHA-1. We then investigate common binary file formats to determine whether they could allow us to construct a malicious SHA-1 for which two *valid* files collide. Finally, we present actual collisions, and characterize the associated instances of malicious SHA-1.

4.1 Constraints

The attack strategy and the available message differences impose several constraints for possible applications. Most importantly, the exact hash function definition with the final constants is only fixed during the attack. This implies that the differences between the two final files will be limited to a single block. In addition, this block must correspond to the first 512 bits of the final files. After this initial collision block, the file pair can be extended arbitrarily with a common suffix. Due to this limitation, for example, the method that was used to find colliding PostScript files for MD5 [12] cannot be applied here.

For the exact values of the first block, the attack allows a certain freedom. The attacker can fix the values of a few bits in advance. However, fixing too many bits will increase the attack complexity. Additionally, choosing the bits is constrained by the fixed message difference. In all our example files, we use message difference MD_1 from Table 3, which offers a slightly better expected attack complexity than MD_2 and MD_3. All of the available message differences have a difference in the first word, as well as the last byte.

4.2 Binary File Format Overview

Binary file formats typically have a predefined structure and in particular a "magic signature" in their first bytes, which is used to identify the type of binary and to define diverse metadata. As a preliminary to the construction of colliding binaries, we provide basic information on binary files so as to understand the obstacles posed for the construction of colliding files.

We also discuss both our failed and successful attempts to build colliding binary executables. Note that once a collision can be created—that is, if the block difference can be introduced without fatally altering the file structure—the programs executed in each of the two colliding files can be fully controlled. In practice, both programs may execute a legitimate application, but one of the two colliding files prepends the execution of a trojan that will persistently compromise the machine.

Magic Signatures. Most binary file formats enforce a "magic signature" at offset 0, to enable programs to recognize the type of file, its version, etc., in order to process it according to its specific format. For example, the utilities `file` and `binwalk` rely mostly on magic signatures to identify files and their type. Some formats, notably most archive formats, also allow the signature to start later in the file, at a higher offset.

Signatures are typically 4 bytes long. Some are longer, such as that of the PNG format (`89504e470d0a1a0a`), or the RAR archive format (`526172211a0700`), and some are smaller (PE's 2-byte "MZ", TIFF's 2-byte "MM" and "II"). Note that none of our colliding blocks offer four unmodified consecutive bytes. This implies that collisions for our malicious SHA-1 cannot be files with a fixed 4-byte signature at offset 0.

Executables: PE. The PE (Portable Executable) format is the standard format for Windows executables (`.exe` files). The PE format, as defined in 1993, is based on the older DOS EXE format (from 1981). PE thus retains the MZ signature (`4d5a`) from DOS, however in PE it is mostly useless: the only components of the header used are the MZ signature and the last component, which is a pointer to the more modern PE header. This leaves an entirely controllable buffer of 58 bytes near the top of the file, which is tempting to use to build colliding PEs.

PE thus seems an interesting candidate for malicious collisions: it is very commonly used, and its header provides freedom degrees to introduce differences. The only restrictions in the header are in the first two bytes (which must be set to the MZ string) and in the four bytes at offset 60, where the 4-byte pointer to the PE header is encoded.

Unfortunately, the structure of the differential attack forces the most significant byte of the PE header to be (at least) `40`. This gives a minimal pointer of `40000000`, that is, 1 GiB. Such a file, even if syntaxically correct, is not supported by Windows: it is correctly parsed, but then the OS fails to load it (In practice, the biggest accepted value for this pointer in a working PE is around `9000000`).

Due to this limitation, we could not construct valid compact PE executables that collide for a malicious SHA-1. Note that the Unix and OS X counterpart of PEs (ELF and Mach-O files, respectively) fix at least the first 4 bytes, and thus cannot be exploited for malicious collisions either.

Headerless Executables: MBR and COM. Some older formats like master boot records (MBR) and DOS executables (COM) do not include a magic signature or any header. Instead, code execution starts directly at offset 0. By introducing a jump instruction to the subsequent block, we can have total control of the first block and thus create collisions (as long as the difference allows for the jump instruction with distinct reasonable addresses). Running in 16-bit x86 code, the block can start with a jump, encoded as `eb XX`, where `XX` is a signed char that should be positive. Both blocks will immediately jump to different pieces of code of colliding MBR or colliding COM. To demonstrate the feasibility of this approach, example files are given in the appendix.

Compressed Archives: RAR and 7z. Like any other archive file format, the RAR archive allows to start at any offset. However, unlike the ZIP, it is parsed top-down. So if a block creates a valid Rar signature that is broken by its twin, then both files can be valid Rars yet different. We could thus create two colliding archives, which can each contain arbitrary content. A very similar method can be used to build colliding 7z archives (and probably other types of compressed archives).

Images: JPEG. The JPEG file format is organized in a chunk-based manner: chunks are called segments, and each segment starts with a 2 bytes marker. The first byte of the marker is always `ff`, the second is anything but `00` or `ff`. A JPEG file must start with a "Start Of Image" (SOI) marker, `ffd8`. Segments have a variable size, encoded directly after the marker, on 2 bytes, in little endian format. Typically, right after the SOI segment starts the APP0 segment, with marker `ffe0`. This segment contains the familiar "JFIF" string.

However, most JPEG viewers do not require the second segment to start right after SOI. Adding megabytes of garbage data between the SOI marker and the APP0 segment of a JPEG file will still make it valid for most tools – as long as this data does not contain any valid marker, `ff(01-fe)`.

Not only we can insert almost-random data before the first segment, but we can insert any dummy segment that has an encoded length – this will enable us to control the parser, to give it the data we want. If each of our colliding files contains a valid segment marker with a different size and offset, each of them can have a valid APP0 segment at a different offset (provided that the sum of segment size and segment offset differs). To make the bruteforcing phase easier, we can use any of the following segments:

1. the APPx segments, with marker `ffe(0-f)`
2. the COM segment, with marker `fffe`

So, after getting 2 colliding blocks, creating 2 JPEG headers with a suitable dummy segment, we can start the actual data of file J_1 after the second dummy segment (with larger sum of segment size and offset). Right after the first dummy segment, we start another dummy segment to cover the actual data of file J_1. After this second dummy segment, the data of the file J_2 can start. If the length of any of the JPEG file cannot fit on 2 bytes, then several dummy segments need to be written consecutively. Thus, we are able to get a colliding pair of valid files, on a modern format, still used daily by most computers.

Combining Formats: Polyglots. Since the formats discussed above require their magical signatures at different positions in the file, it is possible to construct a first block (of 64 bytes) that suits multiple file formats. For instance, JPEG requires fixed values in the first few words, while archives like RAR can start their signature at a higher offset. Thus, we can construct colliding block pairs for a fixed selection of constants that can later be used to construct colliding files of multiple types with almost arbitrary content. Examples are given in Sect. 4.3 (JPEG-RAR) and in the appendix (MBR-RAR-Script).

4.3 Example Files

We use the attack strategy from Sect. 3 to build a colliding pair of JPEG images and one of RAR archives, both for the same set of malicious SHA-1 constants. Since JPEG requires the file to start with `ffd8`, MD_1 is the only one of the message differences given in Table 3 that is suitable for a collision between two JPEG files. The following bytes are set to `ffe?`, where `?` differs between the two files and can be any value. Additionally, the last byte of this first 64-byte-block is fixed to also allow the block to be used for RAR collisions: It is set to the first byte of the RAR signature, `52`, in one of the blocks, and to a different value as determined by MD_1 in the other block. Using these constraints as a starting point, we search for a differential characteristic. The result is given in Table 5 in the appendix. Note that at this point, the first round constant K_1 is fixed to an arbitrary value (we use the original constant), while K_2, K_3, K_4 are still free. They are determined together with the full first 64-byte block in the next phase. The result is the message pair already given in Table 4. The malicious SHA-1

Fig. 2. Colliding JPEG/RAR polyglot file pair for malicious SHA-1 (cf. Table 4).

Table 5. Characteristic corresponding to message difference MD_1, with additional constraints for colliding JPEG/RAR polyglot files (cf. Sect. 4.3).

i	A_i	W_i
0	1001111110001101100110001001010n	111111111011000111111111111000nu
1	00u1101001100--------001111u0n00	11u0001000000000-----0100unu0n00
2	n111n00---1------------uu000un00	u011n1------------------00001000
3	0uuuu111--0---0---uu---0-0un11nn	nnu-n--------------------nn000u1
4	1n01u1110---u-n-------u0011001n0	uu1-u------------------00u0011uu
5	0011011n1n00--0-un0101-10n1u0n00	10u0u--------------------1101u11
6	n1n1n1n01000--1-100101-00n000011	1u111--------------------u-001u0
7	nu1nnnnnnnnnnnnnnnnnnnnnnnn000n1	0n10u00------------------n000nn1
8	101111-10011000000010000111nu0u1	n001u----------------------000u1
9	0-101010100000000000000000001un001	10110100-----------------01u1u00
10	u1n00------------------------01u	1011n---------------------0-00n1
11	-00-0--------------------1100001	u0u----------------------n1n0n00
12	-0010-----------------------00-1	u01-n----------------------100n0
13	----1-----------------------1100	n0100----------------------11011
14	-------------------------------n	u1u-u---------------------0000nn
15	-1--------------------------0---	n1u-u-------------------0un10010
16	-0--------------------------1---	1110----------------------1000un
17	------------------------------u-	nn01---------------------10111u1
18	-------------------------------n	nn001--------------------n-010nu
19	--------------------------------	un1----------------------un111n1
20	--------------------------------	n11-----------------------0011nu
21	------------------------------n-	0u01---------------------01110n1
22	-------------------------------n	0u0----------------------u1100nu
23	--------------------------------	nn1----------------------un001n1
24	--------------------------------	n010-----------------------100un
25	------------------------------n-	1n1---------------------011001n0
26	-n-----------------------------u	011----------------------u-110un
27	--------------------------------	nn00---------------------nu0u0n0
28	------------------------------u-	nn101---------------------11001u
29	n-----------------------------n-	uu1n---------------------n0101n0
30	--------------------------------	1n1u---------------------u1u01u0
31	--------------------------------	uu0u---------------------11101u0
32	-------------------------------u	01n----------------------10110un
33	--------------------------------	01n--------------------10nu00001
34	u-------------------------------	10u----------------------10100nu
35	------------------------------n-	nu0--------------------1001n11n1
36	------------------------------n-	1n0----------------------u0111n0
37	--------------------------------	nun--------------------00u1100u0
38	------------------------------u-	n0u-----------------------110001
39	--------------------------------	10n0-------------------10n010111
40	n-------------------------------	u01----------------------10000n0
41	--------------------------------	101----------------------01u1001
42	u------------------------------u	u10-0---------------------0111un
43	--------------------------------	n0u--------------------01nun0010
44	--------------------------------	n1u----------------------10000nu
45	------------------------------n-	nu0----------------------00111n1
46	------------------------------u-	uuu----------------------u1111u1
47	--------------------------------	uun--------------------10n0000u0
48	------------------------------u-	n00---------------------10101100
49	--------------------------------	100--------------------11n010100
50	------------------------------u-	010--------------------1-0100010
51	--------------------------------	010--------------------01n100010
52	--------------------------------	u01----------------------11100n0
53	--------------------------------	101--------------------010110101
54	-------------------------------n	n11----------------------101100n
55	--------------------------------	u00--------------------110u11000
56	------------------------------u-	100----------------------00011uu
57	------------------------------u-	1u1--------------------11n1011u0
58	-------------------------------u	1u1----------------------n00101n
59	--------------------------------	nu0--------------------10nu001n1
60	--------------------------------	101---------------------110000nu
61	------------------------------n-	un1--------------------0010000n1
62	------------------------------n-	1n--1------------------10u0111n1
63	--------------------------------	uu1--------------------11u0111u0
64	--------------------------------	u10-1------------------0100000n0
65	--------------------------------	0--------------------11110001011
66	------------------------------n-	111-------------------0-000011n1
67	--------------------------------	u1-------------------0110u100100
68	------------------------------u-	--0--------------------011000100
69	--------------------------------	u1---------------------01n000010
70	------------------------------u-	n-1--------------------111011101
71	--------------------------------	-0-------------------1100n01100-
72	------------------------------u-	u----------------------000001110
73	--------------------------------	--1------------------0011n111011
74	------------------------------u-	n-----------------------101111--
75	--------------------------------	---------------------0000n011101
76	--------------------------------	u----------------------1110001u-
77	--------------------------------	-----------------------101000--1
78	--------------------------------	n----------------------00011101-
79	--------------------------------	n---------------------11100101--

variant differs from the standardized SHA-1 in the values of 45 bits of the round constants.

Now, we can append suitable followup blocks to create valid JPEG or RAR file pairs, both with arbitrary content. As an example, both images in Fig. 2 hash to $h(m)$ = `1896b202 394b0aae 54526cfa e72ec5f2 42b1837e` using the malicious round constants K_1 = `5a827999`, K_2 = `4eb9d7f7`, K_3 = `bad18e2f`, K_4 = `d79e5877`.

In a similar fashion, we were able to construct another example block pair for a different set of SHA-1 constants that is suitable for master boot records, shell scripts and RAR archives. All example file pairs and code for verification can be found online at http://malicioussha1.github.io/.

Acknowledgments. The work has been supported by the Austrian Government through the research program FIT-IT Trust in IT Systems (Project SePAG, Project Number 835919).

A Full Characteristic for Malicious SHA-1

Table 5 shows a full differential characteristic corresponding to message difference MD_1 using the notation of generalized conditions of [14]. The message block pair given in Table 4 is based on this differential characteristic, as is the example file pair in Sect. 4.3 that uses this block pair as a first block.

The table shows the message expansion words W_i on the right-hand side and the state words A_i on the left-hand side. Note that the state words $B_i, \ldots, E_i$ can be easily derived from this representation.

The characteristic already specifies the necessary format fragments required in Sect. 4.3: The first 28 bits of word W_0 are set to `ffd8ffe` to accommodate the JPEG format, and the last 8 bits of word W_{15} are fixed as `52` (in one message m, for the RAR header) or `32` (in the other message m^*). Additionally, the first round constant is already fixed to the original value K_1 = `5a827999`, while K_2, K_3, K_4 are still free to be chosen during message modification.

References

1. Adinetz, A.V., Grechnikov, E.A.: Building a collision for 75-round reduced SHA-1 using GPU clusters. In: Kaklamanis, C., Papatheodorou, T., Spirakis, P.G. (eds.) Euro-Par 2012. LNCS, vol. 7484, pp. 933–944. Springer, Heidelberg (2012)
2. Becker, G.T., Regazzoni, F., Paar, C., Burleson, W.P.: Stealthy dopant-level hardware trojans. In: Bertoni, G., Coron, J.-S. (eds.) CHES 2013. LNCS, vol. 8086, pp. 197–214. Springer, Heidelberg (2013)
3. Biham, E.: Cryptanalysis of Patarin's 2-round public key system with S boxes (2R). In: Preneel, B. (ed.) EUROCRYPT 2000. LNCS, vol. 1807, pp. 408–416. Springer, Heidelberg (2000)
4. Biham, E., Carmeli, Y., Shamir, A.: Bug attacks. In: Wagner, D. (ed.) CRYPTO 2008. LNCS, vol. 5157, pp. 221–240. Springer, Heidelberg (2008)

5. Blaze, M., Feigenbaum, J., Leighton, T.: Master key cryptosystems. CRYPTO 1995 rump session (1995). http://www.crypto.com/papers/mkcs.pdf
6. Brassard, G. (ed.): CRYPTO 1989. LNCS, vol. 435. Springer, Heidelberg (1990)
7. 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)
8. Chabaud, F., Joux, A.: Differential collisions in SHA-0. In: Krawczyk, H. (ed.) CRYPTO 1998. LNCS, vol. 1462, pp. 56–71. Springer, Heidelberg (1998)
9. Contini, S., Lenstra, A.K., Steinfeld, R.: VSH, an efficient and provable collision-resistant hash function. In: Vaudenay, S. (ed.) EUROCRYPT 2006. LNCS, vol. 4004, pp. 165–182. Springer, Heidelberg (2006)
10. Damgård, I.: A design principle for hash functions. In: Brassard, G., [6], pp. 416–427
11. Daniel R. L. Brown, S.A.V.: Elliptic curve random number generation. Patent. US 8396213 B2 (2006). http://www.google.com/patents/US8396213
12. Daum, M., Lucks, S.: Hash collisions (the poisoned message attack). CRYPTO 2005 rump session (2005). http://th.informatik.uni-mannheim.de/people/lucks/HashCollisions/
13. De Cannière, C., Mendel, F., Rechberger, C.: Collisions for 70-Step SHA-1: on the full cost of collision search. In: Adams, C., Miri, A., Wiener, M. (eds.) SAC 2007. LNCS, vol. 4876, pp. 56–73. Springer, Heidelberg (2007)
14. De Cannière, C., Rechberger, C.: Finding SHA-1 characteristics: general results and applications. In: Lai, X., Chen, K. (eds.) ASIACRYPT 2006. LNCS, vol. 4284, pp. 1–20. Springer, Heidelberg (2006)
15. Filiol, E.: Malicious cryptography techniques for unreversable (malicious or not) binaries. CoRR abs/1009.4000 (2010)
16. Green, M.: A few more notes on NSA random number generators. Blog post, December 2013. http://blog.cryptographyengineering.com/2013/12/a-few-more-notes-on-nsa-random-number.html
17. Green, M.: The many flaws of Dual_EC_DRBG. Blog post, September 2013. http://blog.cryptographyengineering.com/2013/09/the-many-flaws-of-dualecdrbg.html
18. Johansson, T., Nguyen, P.Q. (eds.): EUROCRYPT 2013. LNCS, vol. 7881. Springer, Heidelberg (2013)
19. Joux, A., Peyrin, T.: Hash functions and the (amplified) boomerang attack. In: Menezes, A. (ed.) CRYPTO 2007. LNCS, vol. 4622, pp. 244–263. Springer, Heidelberg (2007)
20. Lin, L., Kasper, M., Güneysu, T., Paar, C., Burleson, W.: Trojan side-channels: lightweight hardware trojans through side-channel engineering. In: Clavier, C., Gaj, K. (eds.) CHES 2009. LNCS, vol. 5747, pp. 382–395. Springer, Heidelberg (2009)
21. Manuel, S.: Classification and generation of disturbance vectors for collision attacks against SHA-1. Des. Codes Crypt. **59**(1–3), 247–263 (2011)
22. Mendel, F., Nad, T., Schläffer, M.: Finding SHA-2 characteristics: searching through a minefield of contradictions. In: Lee, D.H., Wang, X. (eds.) ASIACRYPT 2011. LNCS, vol. 7073, pp. 288–307. Springer, Heidelberg (2011)
23. Mendel, F., Nad, T., Schläffer, M.: Improving local collisions: new attacks on reduced SHA-256. In: Johansson, T., Nguyen, P.Q., [18], pp. 262–278
24. Menn, J.: Exclusive: secret contract tied NSA and security industry pioneer. Reuters, December 2013. http://www.reuters.com/article/2013/12/20/us-usa-security-rsa-idUSBRE9BJ1C220131220
25. Merkle, R.C.: One way hash functions and DES. In: Brassard, G., [6], pp. 428–446

26. Murdoch, S.J., Lewis, S.: Embedding covert channels into TCP/IP. In: Barni, M., Herrera-Joancomartí, J., Katzenbeisser, S., Pérez-González, F. (eds.) IH 2005. LNCS, vol. 3727, pp. 247–261. Springer, Heidelberg (2005)
27. NIST: Recommendation for random number generation using deterministic random bit generators (revised). NIST Special Publication 800–90 (2007)
28. NIST: Secure hash standard (SHS). FIPS PUB 180–4 (2012)
29. Open crypto audit. http://opencryptoaudit.org. Accessed 28 May 2014
30. Patarin, J., Goubin, L.: Trapdoor one-way permutations and multivariate polynomials. In: Han, Y., Quing, S. (eds.) ICICS 1997. LNCS, vol. 1334, pp. 356–368. Springer, Heidelberg (1997)
31. Pramstaller, N., Rechberger, C., Rijmen, V.: Exploiting coding theory for collision attacks on SHA-1. In: Smart, N.P. (ed.) Cryptography and Coding 2005. LNCS, vol. 3796, pp. 78–95. Springer, Heidelberg (2005)
32. Rijmen, V., Oswald, E.: Update on SHA-1. In: Menezes, A. (ed.) CT-RSA 2005. LNCS, vol. 3376, pp. 58–71. Springer, Heidelberg (2005)
33. Rijmen, V., Preneel, B.: A family of trapdoor ciphers. In: Biham, E. (ed.) FSE 1997. LNCS, vol. 1267, pp. 139–148. Springer, Heidelberg (1997)
34. Schneier, B.: The NSA is breaking most encryption on the internet. Blog post, September 2013. https://www.schneier.com/blog/archives/2013/09/the_nsa_is_brea.html
35. Schoenmakers, B., Sidorenko, A.: Cryptanalysis of the dual elliptic curve pseudorandom generator. Cryptology ePrint Archive, Report 2006/190 (2006)
36. Shah, G., Molina, A., Blaze, M.: Keyboards and covert channels. In: USENIX Security Symposium, pp. 59–75 (2006)
37. Stevens, M.: New collision attacks on SHA-1 based on optimal joint local-collision analysis. In: Johansson, T., Nguyen, P.Q., [18], pp. 245–261
38. Wagner, D., Bionbi, P.: Misimplementation of RC4. Submission for the Third Underhanded C Contest (2007). http://underhanded.xcott.com/?page_id=16
39. Wang, X., Yao, A.C., Yao, F.: Cryptanalysis on SHA-1. NIST - First Cryptographic Hash Workshop, October 31–November 1 (2005). http://csrc.nist.gov/groups/ST/hash/documents/Wang_SHA1-New-Result.pdf
40. Wang, X., Yin, Y.L., Yu, H.: Finding collisions in the full SHA-1. In: Shoup, V. (ed.) CRYPTO 2005. LNCS, vol. 3621, pp. 17–36. Springer, Heidelberg (2005)
41. Wu, H., Bao, F., Deng, R.H., Ye, Q.-Z.: Cryptanalysis of Rijmen-Preneel trapdoor ciphers. In: Ohta, K., Pei, D. (eds.) ASIACRYPT 1998. LNCS, vol. 1514, pp. 126–132. Springer, Heidelberg (1998)
42. Ding-Feng, Y., Kwok-Yan, L., Zong-Duo, D.: Cryptanalysis of 2R schemes. In: Wiener, M. (ed.) CRYPTO 1999. LNCS, vol. 1666, pp. 315–325. Springer, Heidelberg (1999)
43. You broke the internet. http://youbroketheinternet.org. Accessed 28 May 2014
44. Young, A., Yung, M.: Malicious Cryptography: Exposing Cryptovirology. Wiley, Chichester (2004)
45. Young, A., Yung, M.: Monkey: black-box symmetric ciphers designed for MON*opolizing* KEY*s*. In: Vaudenay, S. (ed.) FSE 1998. LNCS, vol. 1372, pp. 122–133. Springer, Heidelberg (1998)
46. Young, A.L., Yung, M.: Backdoor attacks on black-box ciphers exploiting low-entropy plaintexts. In: Safavi-Naini, R., Seberry, J. (eds.) ACISP 2003. LNCS, vol. 2727, pp. 297–311. Springer, Heidelberg (2003)