The first collision for full SHA-1

Marc Stevens[1], Elie Bursztein[2], Pierre Karpman[1], Ange Albertini[2], and Yarik Markov[2]

[1] CWI Amsterdam
[2] Google Research
info@shattered.io
https://shattered.io

Abstract. SHA-1 is a widely used 1995 NIST cryptographic hash function standard that was officially deprecated by NIST in 2011 due to fundamental security weaknesses demonstrated in various analyses and theoretical attacks.
Despite its deprecation, SHA-1 remains widely used in 2017 for document and TLS certificate signatures, and also in many software such as the GIT versioning system for integrity and backup purposes.
A key reason behind the reluctance of many industry players to replace SHA-1 with a safer alternative is the fact that finding an actual collision has seemed to be impractical for the past eleven years due to the high complexity and computational cost of the attack.
In this paper, we demonstrate that SHA-1 collision attacks have finally become practical by providing the first known instance of a collision. Furthermore, the prefix of the colliding messages was carefully chosen so that they allow an attacker to forge two distinct PDF documents with the same SHA-1 hash that display different arbitrarily-chosen visual contents. We were able to find this collision by combining many special cryptanalytic techniques in complex ways and improving upon previous work. In total the computational effort spent is equivalent to $2^{63.1}$ calls to SHA-1's compression function, and took approximately 6 500 CPU years and 100 GPU years. While the computational power spent on this collision is larger than other public cryptanalytic computations, it is still more than 100 000 times faster than a brute force search.

Keywords: Hash function, cryptanalysis, collision attack, collision example, differential path construction.

1 Introduction

A cryptographic hash function $\mathrm{H} : \{0,1\}^* \to \{0,1\}^n$ is a function that computes for any arbitrarily long message M a fixed-length hash value of n bits. It is a versatile cryptographic primitive used in many applications including digital signature schemes, message authentication codes, password hashing and content-addressable storage. The security or even the

proper functioning of many of these applications rely on the assumption that it is practically impossible to *find* collisions, *i.e.* two distinct messages x, y that hash to the same value $\mathrm{H}(x) = \mathrm{H}(y)$. When the hash function behaves in a "sufficently random" way, the expected number of calls to H (or in practice its underlying fixed-size function) to find a collision using an optimal generic algorithm is $\sqrt{\pi/2} \cdot 2^{n/2}$ (see *e.g.* [33, App. A]); an algorithm that is faster at finding collisions for H is then a collision attack for this function.

A major family of hash function is "MD-SHA", which includes MD5, SHA-1 and SHA-2 that all have found widespread use. This family originally started with MD4 [36] in 1990, which was quickly replaced by MD5 [37] in 1992 due to serious attacks [9, 11]. Despite early known weaknesses of its underlying compression function [10], MD5 was widely deployed by the software industry for over a decade. The MD5CRK project that attempted to find a collision for MD5 by brute force was halted early in 2004, when Wang and Yu produced explicit collisions [49], found by a groundbreaking attack that pioneered new techniques. In a major development, Stevens *et al.* [45] later showed that a more powerful type of attack (the so-called *chosen-prefix collision attack*) could be performed against MD5. This eventually led to the forgery of a Rogue Certification Authority that in principle completely undermined HTTPS security [46] in 2008. Despite this, even in 2017 there are still issues in deprecating MD5 for signatures [18].

Currently, the industry is facing a similar challenge in the deprecation of SHA-1, a 1995 NIST standard [31]. It is one of the main hash functions of today, and it also has been facing important attacks since 2005. Based on previous successful cryptanalysis [5, 3, 4] of SHA-0 [30] (SHA-1's predecessor, that only differs by a single rotation in the message expansion function), Wang *et al.* [48] presented in 2005 the very first collision attack on SHA-1 that is faster than brute-force. This attack, while groundbreaking, was purely theoretical as its expected cost of 2^{69} calls to SHA-1's compression function was practically out-of-reach.

Therefore, as a proof of concept, many teams worked on generating collisions for reduced versions of the function: 64 steps [8] (with a cost of 2^{35} SHA-1 calls), 70 steps [7] (cost 2^{44} SHA-1), 73 steps [15] (cost $2^{50.7}$ SHA-1) and finally 75 steps [16] (cost $2^{57.7}$ SHA-1) using extensive GPU computation power.

In 2013, building on these advances and a novel rigorous framework for analyzing SHA-1, the current best collision attack on full SHA-1 was presented by Stevens [43] with an estimated cost of 2^{61} calls to the

SHA-1 compression function. Nevertheless, a publicly known collision still remained out of reach. This was also highlighted by Schneier [38] in 2012, when he estimated the cost of a SHA-1 collision attack to be around US$ 700 K in 2015, down to about US$ 173 K in 2018 (using calculations by Walker based on a 2^{61} attack cost [43], Amazon EC2 spot prices and Moore's Law), which he deemed to be within the resources of criminals.

More recently, a collision for the full compression function underlying SHA-1 was obtained by Stevens *et al.* [44] using a start-from-the-middle approach and a highly efficient GPU framework (first used to mount a similar freestart attack on the function reduced to 76 steps [21]). This required only a reasonable amount of GPU computation power, about 10 days using 64 GPUs, equivalent to approximately $2^{57.5}$ calls to SHA-1 on GPU. Based on this attack, the authors projected that a collision attack on SHA-1 may cost between US$ 75 K and US$ 120 K by renting GPU computing time on Amazon EC2 [39] using spot-instances, which is significantly lower than Schneier's 2012 estimates. These new projections had almost immediate effect when CABForum Ballot 152 to extend issuance of SHA-1 based HTTPS certificates was withdrawn [13], and SHA-1 was deprecated for digital signatures in the IETF's TLS protocol specification version 1.3.

Unfortunately CABForum restrictions on the use of SHA-1 only apply to actively enrolled Certification Authority certificates and not on any other certificates, *e.g.* retracted CA certificates that are still supported by older systems (and CA certificates have indeed been retracted for continued use of SHA-1 certificates to serve to these older systems unchecked by CABForum regulations[1]), and certificates for other TLS applications including up to 10% of credit card payment systems [47, 29]. It thus remains in widespread use across the software industry for, *e.g.*, digital signatures of software, documents, and many other applications, most notably in the GIT versioning system.

It is well worth noting that academic researchers have not been the only ones to compute (and exploit) hash function collisions. Nation-state actors [34, 25, 24] have been linked to the highly advanced espionage malware "Flame" that was found targeting the Middle-East in May 2012. As it turned out, it used a forged signature to infect Windows machines via a man-in-the-middle attack on *Windows Update*. Using a new technique of *counter-cryptanalysis* that is able to expose cryptanalytic collision attacks given only one message from a colliding message pair, it was proven that

[1] For instance, SHA-1 certificates are still being sold by CloudFlare at the time of writing: https://www.cloudflare.com/ssl/dedicated-certificates/

the forged signature was made possible by a *then secret* chosen-prefix attack on MD5 [42, 12].

2 Our contributions

Table 1: Colliding message blocks for SHA-1.

CV_0	4e	a9	62	69	7c	87	6e	26	74	d1	07	f0	fe	c6	79	84	14	f5	bf	45
$M_1^{(1)}$			7f	46	dc	93	a6	b6	7e	01	3b	02	9a	aa	1d	b2	56	0b		
			45	ca	67	d6	88	c7	f8	4b	8c	4c	79	1f	e0	2b	3d	f6		
			14	f8	6d	b1	69	09	01	c5	6b	45	c1	53	0a	fe	df	b7		
			60	38	e9	72	72	2f	e7	ad	72	8f	0e	49	04	e0	46	c2		
$CV_1^{(1)}$	8d	64	d6	17	ff	ed	53	52	eb	c8	59	15	5e	c7	eb	34	f3	8a	5a	7b
$M_2^{(1)}$			30	57	0f	e9	d4	13	98	ab	e1	2e	f5	bc	94	2b	e3	35		
			42	a4	80	2d	98	b5	d7	0f	2a	33	2e	c3	7f	ac	35	14		
			e7	4d	dc	0f	2c	c1	a8	74	cd	0c	78	30	5a	21	56	64		
			61	30	97	89	60	6b	d0	bf	3f	98	cd	a8	04	46	29	a1		
CV_2	1e	ac	b2	5e	d5	97	0d	10	f1	73	69	63	57	71	bc	3a	17	b4	8a	c5

CV_0	4e	a9	62	69	7c	87	6e	26	74	d1	07	f0	fe	c6	79	84	14	f5	bf	45
$M_1^{(2)}$			73	46	dc	91	66	b6	7e	11	8f	02	9a	b6	21	b2	56	0f		
			f9	ca	67	cc	a8	c7	f8	5b	a8	4c	79	03	0c	2b	3d	e2		
			18	f8	6d	b3	a9	09	01	d5	df	45	c1	4f	26	fe	df	b3		
			dc	38	e9	6a	c2	2f	e7	bd	72	8f	0e	45	bc	e0	46	d2		
$CV_1^{(2)}$	8d	64	c8	21	ff	ed	52	e2	eb	c8	59	15	5e	c7	eb	36	73	8a	5a	7b
$M_2^{(2)}$			3c	57	0f	eb	14	13	98	bb	55	2e	f5	a0	a8	2b	e3	31		
			fe	a4	80	37	b8	b5	d7	1f	0e	33	2e	df	93	ac	35	00		
			eb	4d	dc	0d	ec	c1	a8	64	79	0c	78	2c	76	21	56	60		
			dd	30	97	91	d0	6b	d0	af	3f	98	cd	a4	bc	46	29	b1		
CV_2	1e	ac	b2	5e	d5	97	0d	10	f1	73	69	63	57	71	bc	3a	17	b4	8a	c5

We are the first to exhibit an example collision for SHA-1, presented in Table 1, thereby proving that theoretical attacks on SHA-1 have now become practical. Our work builds upon the best known theoretical collision attack [43] with estimated cost of 2^{61} SHA-1 calls. This is an *identical-prefix collision attack*, where a given prefix P is extended with two distinct *near-collision block pairs* such that they collide for any suffix S:

$$\text{SHA-1}\left(P\|M_1^{(1)}\|M_2^{(1)}\|S\right) = \text{SHA-1}\left(P\|M_1^{(2)}\|M_2^{(2)}\|S\right). \qquad (1)$$

Table 2: Identical prefix of our collision.

```
25 50 44 46 2d 31 2e 33 0a 25 e2 e3 cf d3 0a 0a | %PDF-1.3.%......
0a 31 20 30 20 6f 62 6a 0a 3c 3c 2f 57 69 64 74 | .1 0 obj.<</Widt
68 20 32 20 30 20 52 2f 48 65 69 67 68 74 20 33 | h 2 0 R/Height 3
20 30 20 52 2f 54 79 70 65 20 34 20 30 20 52 2f |  0 R/Type 4 0 R/
53 75 62 74 79 70 65 20 35 20 30 20 52 2f 46 69 | Subtype 5 0 R/Fi
6c 74 65 72 20 36 20 30 20 52 2f 43 6f 6c 6f 72 | lter 6 0 R/Color
53 70 61 63 65 20 37 20 30 20 52 2f 4c 65 6e 67 | Space 7 0 R/Leng
74 68 20 38 20 30 20 52 2f 42 69 74 73 50 65 72 | th 8 0 R/BitsPer
43 6f 6d 70 6f 6e 65 6e 74 20 38 3e 3e 0a 73 74 | Component 8>>.st
72 65 61 6d 0a ff d8 ff fe 00 24 53 48 41 2d 31 | ream......$SHA-1
20 69 73 20 64 65 61 64 21 21 21 21 21 85 2f ec |  is dead!!!!!./.
09 23 39 75 9c 39 b1 a1 c6 3c 4c 97 e1 ff fe 01 | .#9u.9...<L.....
```

The computational effort spent on our attack is estimated to be equivalent to $2^{63.1}$ SHA-1 calls (see Section 6). There is certainly a gap between the theoretical attack as presented in [43] and our executed practical attack that was based on it. Indeed, the theoretical attack's estimated complexity does not include the inherent relative loss in efficiency when using GPUs, nor the inefficiency we encountered in actually launching a large scale computation distributed over several data centers. Moreover, the construction of the second part of the attack was significantly more complicated than could be expected from the literature.

To find the first near-collision block pair $(M_1^{(1)}, M_1^{(2)})$ we employed the open-source code from [43], which was modified to work with our prefix P given in Table 2, and for large scale distribution over several data centers. To find the second near-collision block pair $(M_2^{(1)}, M_2^{(2)})$ that leads to the collision was more challenging, as the attack cost is known to be significantly higher, but also because of additional obstacles.

In Section 5 we will discuss in particular the process of building the second near-collision attack. Essentially we followed the same steps as was done for the first near-collision attack [43], combining many existing cryptanalytic techniques. Yet we further employed the SHA-1 collision search GPU framework from Karpman *et al.* [21] to achieve a significantly more cost efficient attack.

We also describe two new additional techniques used in the construction of the second near-collision attack. The first allowed us to use additional differential paths around step 23 for increased success probability and more degrees of freedom without compromising the use of an early-stop

technique. The second was necessary to overcome a serious problem of an unsolvable strongly over-defined system of equations over the first few steps of SHA-1's compression function that threatened the feasibility of finishing this project.

As can be deduced from Equation 1, our example colliding files only differ in two successive random-looking message blocks generated by our attack. We exploit these limited differences to craft two colliding PDF documents containing arbitrary distinct images. Examples can be downloaded from `https://shattered.io`. PDFs with the same MD5 hash have previously been constructed by Gebhardt *et al.* [14] by exploiting so-called Indexed Color Tables and Color Transformation functions. However, this method is not effective for many common PDF viewers that lack support for these functionalities. Our PDFs rely on distinct parsings of JPEG images, similar to Gebhardt *et al.*'s TIFF technique [14] and Albertini *et al.*'s JPEG technique [1]. Yet we improved upon these basic techniques using very low-level "wizard" JPEG features such that these work in all common PDF viewers, and even allow very large JPEGs that can be used to craft multi-page PDFs. This overall approach and the technical details will be described in a separate article [2].

The remainder of this paper is organized as follows. We first give a brief description of SHA-1 in Section 3. Then, we give a high-level overview of our attack in Section 4, followed by Section 5 that details the entire process and the cryptanalytic techniques employed, where we also highlight improvements with respect to previous work. Finally, we discuss the large-scale distributed computations required to find the two near-collision block pairs in Section 6. The parameters used to find the second colliding block are given in the appendix, in Section A.

3 The SHA-1 hash function

We provide a brief description of SHA-1 as defined by NIST [31]. SHA-1 takes an arbitrary-length message and computes a 160-bit hash. It divides the (padded) input message into k blocks $M_1, \ldots, M_k$ of 512 bits. The 160-bit internal state CV_j of SHA-1, called the chaining value, is initialized to a predefined initial value $CV_0 = IV$. Each message block is then fed to a compression function h that updates the chaining value, *i.e.* $CV_{j+1} = h(CV_j, M_{j+1})$, for $0 \leq j < k$, where the final CV_k is output as the hash.

The compression function h takes a 160-bit chaining value CV_j and a 512-bit message block M_{j+1} as inputs, and outputs a new 160-bit chaining value CV_{j+1}. It mixes the message block into the chaining value as follows,

operating on *words*, simultaneously seen as 32-bit strings and as elements of $\mathbb{Z}/2^{32}\mathbb{Z}$: the input chaining value is parsed as five words a, b, c, d, e, and the message block as 16 words $m_0, \ldots, m_{15}$. The latter are expanded into 80 words using the following recursive linear equation:

$$m_i = (m_{i-3} \oplus m_{i-8} \oplus m_{i-14} \oplus m_{i-16})^{\circlearrowleft 1}, \quad \text{for } 16 \le i < 80.$$

Starting from $(A_{-4}, A_{-3}, A_{-2}, A_{-1}, A_0) := (e^{\circlearrowleft 2}, d^{\circlearrowleft 2}, c^{\circlearrowleft 2}, b, a)$, each m_i is mixed into an intermediate state over 80 steps $i = 0, \ldots, 79$:

$$A_{i+1} = A_i^{\circlearrowleft 5} + \varphi_i(A_{i-1}, A_{i-2}^{\circlearrowleft 2}, A_{i-3}^{\circlearrowleft 2}) + A_{i-4}^{\circlearrowleft 2} + K_i + m_i,$$

where φ_i and K_i are predefined Boolean functions and constants:

step i	$\varphi_i(x, y, z)$	K_i
$0 \le i < 20$	$\varphi_{\text{IF}} = (x \wedge y) \vee (\neg x \wedge z)$	`0x5a827999`
$20 \le i < 40$	$\varphi_{\text{XOR}} = x \oplus y \oplus z$	`0x6ed9eba1`
$40 \le i < 60$	$\varphi_{\text{MAJ}} = (x \wedge y) \vee (x \wedge z) \vee (y \wedge z)$	`0x8f1bbcdc`
$60 \le i < 80$	$\varphi_{\text{XOR}} = x \oplus y \oplus z$	`0xca62c1d6`

After the 80 steps, the new chaining value is computed as the sum of the input chaining value and the final intermediate state:

$$CV_{j+1} = (a + A_{80}, b + A_{79}, c + A_{78}^{\circlearrowleft 2}, d + A_{77}^{\circlearrowleft 2}, e + A_{76}^{\circlearrowleft 2}).$$

4 Overview of our SHA-1 collision attack

We illustrate our attack from a high level in Figure 1. Starting from identical chaining values for two messages, we use two pairs of blocks. The differences in the first block pair cause a small difference in the output chaining value, which is canceled by the difference in the second block pair, leading again to identical chaining values and hence a collision (indicated by (2)). We employ *differential paths* that are a precise description of differences in state words and message words and of how these differences should propagate through the 80 steps.

Note that although the first five state words are fixed by the chaining value, one can freely modify message words and thus directly influence the next sixteen state words. Moreover, with additional effort this can be extended to obtain limited influence over another eight state words. However, control over the remaining state words (indicated by (1)) is very hard and thus requires very sparse target differences that correctly

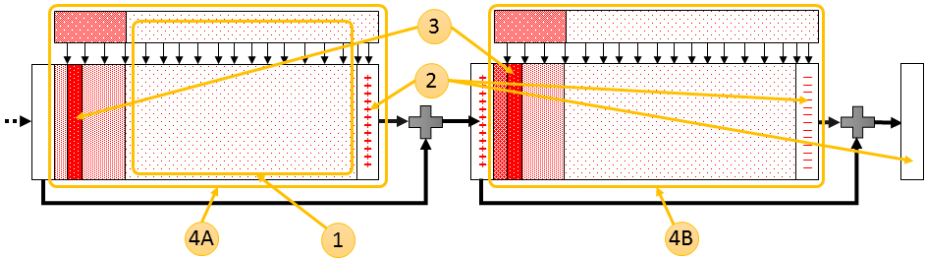

Fig. 1: Attack overview

propagate with probability as high as possible. Furthermore, these need to be compatible with differences in the expanded message words. The key solution is the concept of *local collisions* [5], where any state bit-difference introduced by a perturbation message bit-difference is to be canceled in the next five steps using correction message bit-differences.

To ensure all message word bit differences are compatible with the linear message expansion, one uses a *disturbance vector* (DV) [5] that is a correctly expanded message itself, but where every "1" bit marks the start of a local collision. The selection of a good disturbance vector has a very high impact on the overall attack cost. As previously shown by Wang *et al.* [48], the main reason of using two block pairs (*i.e.* to search for a near-collision over a first message block, that is completed to a full collision over a second) instead of only one is that this choice alleviates an important restriction on the disturbance vector, namely that there are no state differences after the last step. Similarly, it may be impossible to unite the input chaining value difference with the local collisions for an arbitrary disturbance vector. This was solved by Wang *et al.* [48] by crafting a tailored differential path (called the non-linear (NL) path, indicated by (3)) that over the first 16 steps connects the input chaining value differences to the local collision differences over the remaining steps (called the linear path, referring to the linear message expansion dictating the local collision positions).

One has to choose a good disturbance vector, then craft a non-linear differential path for each of the two near-collision attacks (over the first and second message blocks), determine a system of equations over all steps and finally find a solution in the form of a message block pair (as indicated by (4A) and (4B)). Note that one can only craft the non-linear path for the second near-collision attack once the chaining values resulting

from the first block pair are known. This entire process including our improvements is described below.

5 Near-collision attack procedure

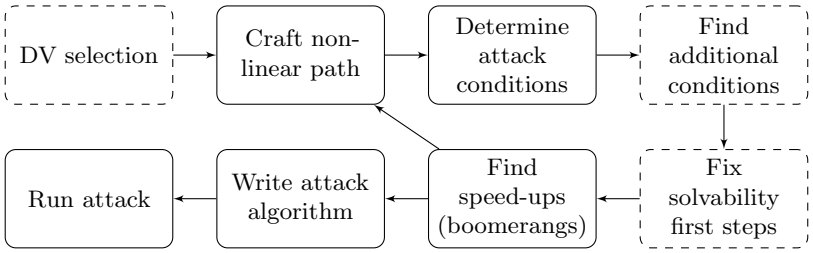

Fig. 2: The main steps for each near-collision attack.

This section describes the overall procedure of each of the two near-collision attacks. Since we relied on our modification of Stevens' public source-code [43, 17] for the first near-collision attack, we focus on our extended procedure for our second near-collision attack. As shown in Figure 2, this involves the following steps that are further detailed below:

1. selection of the disturbance vector (same for both attacks);
2. construction of the non-linear differential path;
3. determine attack conditions over all steps;
4. find additional conditions beyond the fixed differential path for early-stop;
5. if necessary fix solvability of attack conditions over the first few steps;
6. find message modification rules to speed-up collision search;
7. write the attack algorithm;
8. finally, run the attack to find a near-collision block pair.

5.1 Disturbance Vector selection

The selection of which disturbance vector to use is a major choice, as it directly determines many aspects of the collision attack. These include the message XOR differences, but also in theory the optimal attack choices over the linear path, including the optimal set of candidate endings for the non-linear path together with optimal linear message-bit equations that maximize the success probability over the linear part.

Historically several approaches have been used to analyze a disturbance vector to estimate attack costs over the linear part. Initially, the Hamming weight of the DV that counts the active number of local collisions was used (see *e.g.* [4, 35]). For the first theoretical attack on SHA-1 with cost 2^{69} SHA-1-calls by Wang *et al.* [48] a more refined measure was used, that counts the number of bit-conditions on the state and message bits that ensure that the differential path would be followed. This was later refined by Yajima *et al.* [51] to a more precise count by exploiting all possible so-called bit compressions and interactions through the Boolean functions. However, this approach does not allow any difference in the carry propagation, which otherwise could result in alternate differential paths that may improve the overall success probability. Therefore, Mendel *et al.* [28] proposed to use the more accurate probability of single local collisions where carry propagations are allowed, in combination with known local collision interaction corrections.

The current state-of-the-art is joint-local-collision analysis (JLCA) introduced by Stevens [43, 41] which given sets of allowed differences for each state word A_i and message word m_i (given by the disturbance vector) computes the exact optimal success probability over the specified steps by exhaustively evaluating all differential paths with those allowed differences. This approach is very powerful as it also provides important information for the next steps, namely the set of optimal chaining value differences (by considering arbitrary high probability differences for the last five A_is) and the set of optimal endings for the non-linear path, together with a corresponding set of message-bit equations, using which the optimal highest success probability of the specified steps can actually be achieved. The best theoretical collision attack on SHA-1 with cost 2^{61} SHA-1 calls [43] was built using this analysis. As we build upon this collision attack, we use the same disturbance vector, named $\mathrm{II}(52, 0)$ by Manuel [26] and originally described by Jutla and Patthak [20].

5.2 Construction of a non-linear differential path

Once the disturbance vector and the corresponding linear part of the differential path have been fixed, the next step consists in finding a suitable non-linear path connecting the chaining value pair (with fixed differences) to the linear part. This step needs to be done separately for each near-collision attack of the full collision attack[2].

[2] We eventually produced two message block pair solutions for the first near-collision attack. This provided a small additional amount of freedom in the search for the non-linear path of the second block.

As explained for instance in [43], in the case of the first near-collision attack, the attacker has the advantage of two additional freedoms. Firstly, an arbitrary prefix can be included before the start of the attack to prefulfill a limited number of conditions on the chaining value. This allows greater freedom in constructing the non-linear path as this does not have to be restricted to a specific value of the chaining value pair, whereas the non-linear path for the second near-collision attack has to start from the specific given value of input chaining value pair. Secondly, it can use the entire set of output chaining value differences with the same highest probability. The first near-collision attack is not limited to a particular value and succeeds when it finds a chaining value difference in this set, whereas the second near-collision attack has to cancel the specific difference in the resulting chaining value pair. Theory predicts the first near-collision attack to be at least a factor six faster than the second attack [43]. For our collision attack it is indeed the second near-collision attack that dominates the overall attack complexity.

Historically, the first non-linear paths for SHA-1 were hand-crafted by Wang *et al.*. Several algorithms were subsequently developed to automatically search for non-linear paths for MD5, SHA-1, and other functions of the MD-SHA family. The first automatic search for SHA-1 by De Cannière and Rechberger [8] was based on a guess-and-determine approach. This approach tracks the allowed values of each bit pair in the two related compression function computations. It starts with no constraints on the values of these bit pairs other than the chaining value pair and the linear part differences. It then repeatedly restricts values on a selected bit pair and then propagates this information via the step function and linear message expansion relation, *i.e.*, it determines and eliminates previously-allowed values for other bit pairs that are now impossible due the added restriction. Whenever a contradiction occurs, the algorithm backtracks and chooses a different restriction on the last selected bit pair.

Another algorithm for SHA-1 was introduced by Yajima *et al.* [52] that is based on a meet-in-the-middle approach. It starts from two fully-specified differential paths; the first is obtained from a forward expansion of the input chaining value pair, whereas the other is obtained from a backward expansion of the linear path. It then tries to connect these two differential paths over the remaining five steps in the middle by recursively iterating over all solutions over a particular step.

A similar meet-in-the-middle algorithm was independently first developed for MD5 and then adapted to SHA-1 by Stevens *et al.* [45, 41, 17], which operates on bit-slices and is more efficient. The open-source Hash-

Clash project [17] seems to be the only publicly available non-linear path construction implementation, which we improved as follows. Originally, it expanded a large set of differential paths step by step, keeping only the best N paths after each step, for some user-specified number N. However, there might be several good differential paths that result in the same differences and conditions around the connecting five steps, where either none or all lead to fully-connected differential paths. Since we only need the best fully-connected differential path we can find, we only need to keep a best differential path from each subset of paths with the same differences and conditions over the last five steps that were extended. So to remove this redundancy, for each step we extend and keep, say, the $4N$ best paths, then we remove all such superfluous paths, and finally keep at most N paths. This improvement led to a small but very welcome reduction in the amount of differential path conditions under the same path construction parameter choices, but also allowed a better positioning of the largest density of sufficient conditions for the differential path.

Construction of a very good non-linear path for the second near-collision attack using our improved HashClash version took a small effort with our improvements, yet even allowed us to restrict the section with high density of conditions to just the first six steps. However, to find a very good non-linear differential path that is also solvable turned out to be more complicated. Our final solution is described in Section 5.5, which in the end did allow us to build our attack on the best non-linear path we found without any compromises. The fixed version of this best non-linear path is presented in Figure 3, Section A.

5.3 Determine attack conditions

Having selected the disturbance vector and constructed a non-linear path that bridges into the linear part, the next step is to determine the entire system of equations for the attack. This system of equations is expressed entirely over the computation of message $M^{(1)}$, and not over $M^{(2)}$, and consists of two types of equations:

1. Linear equations over message bits. These are used to control the additive signs of the message word XOR differences implied by the disturbance vector. Since there are many different "signings" over the linear part with the same highest probability, instead of one specific choice one uses a linear hull that captures many choices to reduce the amount of necessary equations.

2. Linear equations over state bits given by a fixed differential path up to some step i (that includes the non-linear path). These control whether there is a difference in a state bit and which sign it has, furthermore they force target differences in the outputs of the Boolean functions φ_i.

We determine this entire system by employing our implementation of joint-local-collision analysis that has been improved as follows. JLCA takes input sets of allowed differences for each A_i and m_i and exhaustively analyzes the set of differential paths with those allowed differences, which originally is only used to analyze the linear part. We additionally provide it with specific differences for A_i and m_i as given by the non-linear path, so we can run JLCA over all 80 steps and have it output an optimal fixed differential path over steps $0, \ldots, 22$ together with an optimal set of linear equations over message bits over the remaining steps. These are optimal results since JLCA guarantees these lead to the highest probability that is possible using the given allowed differences, but furthermore that a largest linear hull is used to minimize the amount of equations.

Note that having a fixed differential path over more steps directly provides more state bit equations which is helpful in the actual collision search because we can apply an early-stop technique. However, this also adds further restrictions on A_i limiting a set of allowed differences to a single specific difference. In our case limiting A_{24} would result, besides a drop in degrees of freedom, in a lower overall probability, thus we only use a fixed differential path up to step 22, *i.e.*, up to A_{23}. Below in Section 5.4 we show how we compensated for fewer state equations that the actual collision search uses to early stop.

5.4 Find additional state conditions

As explained in Section 5.3, the system of equations consists of linear equations over (expanded) message bits and linear equations over state bits. In the actual collision search algorithm, we depend on these state bit equations to stop computation on a bad current solution as early as possible and start backtracking. These state bit equations are directly given by a fixed differential path, where every bit difference in the state and message is fixed. Starting from step 23 we allow several alternate differential paths that increase success probability, but also allow distinct message word differences that lead to a decrease in the overall number of equations. Each alternate differential path depends on its own (distinct) message word differences and leads to its own state bit equations. To

find additional equations, we also consider linear equations over state *and* message bits around steps 21–25. Although in theory these could be computed by JLCA by exhaustively reconstructing all alternate differential paths and then determining the desired linear equations, we instead took a much simpler approach. We generated a large amount of random solutions of the system of equations up to step 31 using an unoptimized general collision search algorithm. We then proceeded to exhaustively test potential linear equations over at most four state bits and message bits around steps 21–25, which is quite efficient as on average only two samples needed to be checked for each bad candidate. The additional equations we found and used for the collision search are shown in Table 4, Section A.

5.5 Fix solvability over the first steps

This step is not required when there are sufficient degrees of freedom in the non-linear part, as was the case in the first-block near-collision attack. As already noted, in the case of the second-block near-collision attack, the non-linear path has to start will a fully-fixed chaining value and has significantly more conditions in the first steps. As a result, the construction of a very good *and* solvable non-linear differential path for the second near-collision attack turned out to be quite complex. Our initially constructed paths unfortunately proved to be unsolvable over the first few steps. We tried several approaches including using the guess-and-determine non-linear path construction to make corrections as done by Karpman *et al.* [21], as well as using worse differential path construction parameters, but all these attempts led to results that not only were unsatisfactory but that even threatened the feasibility of the second near-collision attack. Specifically, both approaches led to differential paths with a significantly increased number of conditions, bringing the total number of degrees of freedom critically low. Moreover, the additional conditions easily conflicted with candidate speed-up measures named "boomerangs" necessary to bring the attack's complexity down to a feasible level. Our final solution was to encode this problem into a satisfiability (SAT) problem and use a SAT solver to find a drop-in replacement differential path over the first eight steps that is solvable.

More specifically, we adapted the SHA-1 SAT system generator from Nossum[3] [32] (initially used to compute reduced-round practical preimages) to generate two independent 8-step compression function computations, which we then linked by adding constraints that set the given input

[3] https://github.com/vegard/sha1-sat

chaining value pair, the message XOR differences over $m_0, \ldots, m_7$, the path differences of $A_4, \ldots, A_8$ and the path conditions of $A_5, \ldots, A_8$. In effect, we allowed complete freedom over A_1, A_2, A_3 and some freedom over A_4. All solutions were exhaustively generated by MiniSAT[4] and then converted into drop-in replacement paths, from which we kept the one with fewest conditions.

This allowed us to build our attack on the best non-linear path we found without any compromises and the corrected non-linear path is presented in Figure 3, Section A. Note that indeed the system of equations is over-defined: over the first five steps, there are only 15 state bits without an equation, while at the same time there are 23 message equations.

5.6 Find message modifications to speed-up collision search

To speed-up the collision search significantly, it is important to employ message modification rules, that make small changes in the current message block that do not affect any bit involved with the state and message-bit equations up to some step n (with sufficiently high probability). This effectively allows such a message modification rule to be applied to one solution up to step n to generate several solutions up to the same step with almost no additional cost, thereby significantly reducing the average cost to generate solutions up to step n.

The first such speed-up technique that was developed in attacks of the MD-SHA family was called *neutral bits*, introduced by Biham and Chen to improve attacks on SHA-0 [3]. A message bit is *neutral* up to a step n if flipping this bit causes changes that do not interact with differential path conditions up to step n with high probability. As the diffusion of SHA-0/SHA-1's step function is rather slow, it is not hard to find many bits that are neutral for a few steps.

A nice improvement of the original neutral bits technique was ultimately described by Joux and Peyrin as "boomerangs" [19]. It consists in carefully selecting a few bits that are all flipped together in such a way that this effectively flips, say, only one state bit in the first 16 steps, and such that the diffusion of uncontrollable changes is significantly delayed. This idea can be instantiated efficiently by flipping together bits that form a local collision for the step function. This local collision will eventually introduce uncontrollable differences through the message expansion; however, these do not appear immediately, and if all conditions for the local collision to be successful are verified, the first few steps after the

[4] http://minisat.se/

introduction of its initial perturbation will be free of any difference. Joux and Peyrin then noted that sufficient conditions for the local collision can be pre-satisfied when creating the initial partial solution, effectively leading to probability-one local collisions. This leads to a few powerful message modification rules that are neutral up to very late steps.

A closely-related variant of boomerangs is named *advanced message modification* by Wang *et al.* in their attack of the MD-SHA family (see *e.g.* [48]). While the objective of this technique is also to exploit the available freedom in the message, it applies this in a distinct way by identifying ways of interacting with an isolated differential path condition with high probability. Then, if an initial message pair fails to verify a condition for which a message modification exists, the bits of the latter are flipped, so that the resulting message pair now verifies the condition with high probability.

In our attack, we used both neutral bits and boomerangs as message modification rules. This choice was particularly motivated by the ability to efficiently implement these speed-up techniques on GPUs, used to compute the second block of the collision, similar to [21, 44].

Our search process for finding the neutral bits follows the one described in [44]. Potential boomerangs are selected first, one being eligible if its initial perturbation does not interact with differential path conditions and if the corrections of the local collision do not break some linear message-bit-relation (this would typically happen if an odd number of bits to be flipped are part of such a relation). The probability with which a boomerang eventually interacts with path conditions is then evaluated experimentally by activating it on about 4 000 independent partial solutions; the probability threshold used to determine up to which step a boomerang can be used is set to 0.9, meaning that it can be used to generate an additional partial solution at step n from an existing one if it does not interact with path conditions up to step n with probability more than 0.1. Once boomerangs have been selected, the sufficient conditions necessary to ensure that their corresponding local collisions occur with probability 1 are added to the differential path, and all remaining free message bits are tested for neutrality using the same process (*i.e.*, a bit is only eligible if flipping it does not trivially violate path conditions or make it impossible to later satisfy message-bit-relations, and its quality is evaluated experimentally).

The list of neutral bits and boomerangs used for the second block of the attack is given in Section A. There are 51 neutral bits, located on

message words m_{11} to m_{15}, and three boomerangs each made of a single local collision started on m_6 (for two of them) or m_9.

5.7 Attack implementation

A final step in the design of the attack is to implement it. This is needed for obvious reasons if the goal is to find an actual collision as we do here, but it is also a necessary step if one wishes to obtain a precise estimate of the complexity of the attack. Indeed, while the complexity of the probabilistic phase of the attack can be accurately computed using JLCA (or can also be experimentally determined by sampling many mock partial solutions), there is much more uncertainty as to "where" this phase actually starts. In other words, it is hard to exactly predict how effective the speed-up techniques can be without actually implementing them. The only way to determine the real complexity of an attack is then to implement it, measure the rate of production of partial solutions up to a step where there is no difference in the differential path for five consecutive state words, and use JLCA to compute the exact probability of obtaining a (near-)collision over the remaining steps.

The first near-collision block pair of the attack was computed with CPUs, using an adapted version of the HashClash software [17]. As the original code was not suitable to run on a large scale, a significant effort was spent to make it efficient on the hundreds of cores necessary to obtain a near-collision in reasonable time. The more expensive computation of the second block was done on GPUs, based on the framework used by Karpman *et al.* [21], which we briefly describe below.

The main structure used in this framework consists in first generating *base solutions* on CPUs that fix the sixteen free message words, and then to use GPUs to extend these to partial solutions up to a late step, by only exploiting the freedom offered by speed-up techniques (in particular neutral bits and boomerangs). These partial solutions are then sent back to a CPU to check if they result in collisions.

The main technical difficulty of this approach is to make the best use of the power offered by GPUs. Notably, their programming model differs from the one of CPUs in how diverse the computations run on their many available cores can be: on a multicore CPU, every core can be used to run an independent process; however, even if a recent GPU can feature many more cores than a CPU (for instance, the Nvidia GTX 970 used in [21, 44] and the initial implementation of this attack features 1664 cores), they can only be programmed at the granularity of *warps* made of 32 threads, which must then run the same code. Furthermore,

divergence in the control flow of threads of a single warp is dealt with by serializing the diverging computations; for instance, if a single thread takes a different branch than the rest of the warp in an *if* statement, all the other threads become idle while it is taking its own branch. This limitation would make a naïve parallel implementation of the usage of neutral bits rather inefficient, and there is instead a strong incentive to minimize control-flow divergence when implementing the attack.

The approach taken by Karpman *et al.* [21] to limit the impact of the inherent divergence in neutral bit usage is to decompose the attack process step by step and to use the fair amount of memory available on recent GPUs to store partial solutions up to many different steps in shared buffers. In a nutshell, all threads of a single warp are asked to load their own partial solution up to a certain state word A_i, and they will together apply all neutral bits available at this step, each time checking if the solution can be validly extended to a solution up to A_{i+1}; if and only if this is the case, this solution is stored in the buffer for partial solutions up to A_{i+1}, and this selective writing operation is the only moment where the control flow of the warps may diverge.

To compute the second block pair of the attack, and hence obtain a full collision, we first generated base solutions consisting of partial solutions up to A_{14} on CPU, and used GPUs to generate additional partial solutions up to A_{26}. These were further probabilistically extended to partial solutions up to A_{53}, still using GPUs, and checking whether they resulted in a collision was finally done on a CPU. The probability of such a partial solution to also lead to a collision can be computed by JLCA to be equal to $2^{-27.8}$, and $2^{-48.7}$ for partial solutions up to A_{33} (these probabilities could in fact both be reduced by a factor $2^{0.6}$; however, the ones indicated here correspond to the attack we carried out). On a GTX 970, a prototype implementation of the attack produced partial solutions up to A_{33} at a rate of approximately 58 100 per second, while the full SHA-1 compression function can be evaluated about $2^{31.8}$ times per second on the same GPU. Thus, our attack has an expected complexity of $2^{64.7}$ on this platform.

Finally, adapting the prototype GPU implementation to a large-scale infrastructure suitable to run such an expensive computation also required a fair amount of work.

6 Computation of the collision

This section gives some details about the computation of the collision and provides a few comparisons with notable cryptographic computations.

6.1 Units of complexity

The complexity figures given in this section follow the common practice in the cryptanalysis of symmetric schemes of comparing the efficiency of an attack to the cost of using a generic algorithm achieving the same result. This can be made by comparing the time needed, with the same resources, to *e.g.* compute a collision on a hash function by using a (memoryless) generic collision search *versus* by using a dedicated process. This comparison is usually expressed by dividing the time taken by the attack, *e.g.* in core hours, by the time taken to compute the attacked primitive once on the same platform; the cost of using a generic algorithm is then left implicit. This is for instance how the figure of $2^{64.7}$ from Section 5.7 has been derived.

While this approach is reasonable, it is far from being as precise as what a number such as $2^{64.7}$ seems to imply. We discuss below a few of its limitations.

The impact of code optimization. An experimental evaluation of the complexity of an attack is bound to be sensitive to the quality of the implementation, both of the attack itself and of the reference primitive used as a comparison. A hash function such as SHA-1 is easy to implement relatively efficiently, and the difference in performance between a reference and optimized implementation is likely to be small. This may however not be true for the implementation of an attack, which may have a more complex structure. A better implementation may then decrease the "complexity" of an attack without any cryptanalytical improvements.

Although we implemented our attack in the best way we could, one cannot exclude that a different approach or some modest further optimizations may lead to an improvement. However, barring a radical redesign, the associated gain should not be significant; the improvements brought by some of our own low-level optimizations was typically of about 15%.

The impact of the attack platform. The choice of the platform used to run the attack may have a more significant impact on its evaluated complexity. While a CPU is by definition suitable to run general-purpose computations, this is not the case of *e.g.* GPUs. Thus, the gap between how fast a simple computation, such as evaluating the compression function of SHA-1, and a more complex one, such as our attack, need not be the same on the two kinds of architectures. For instance, the authors of [21] noticed that their 76-step freestart attack could be implemented on CPU (a 3.2 GHz Haswell Core i5) for a cost equivalent to $2^{49.1}$ compression function

computations, while this increased to $2^{50.25}$ on their best-performing GTX 970, and $2^{50.34}$ on average.

This difference leads to a slight paradox: from an attacker's point of view, it may seem best to implement the attack on a CPU in order to be able to claim a better attack complexity. However, a GPU being far more powerful, it is actually much more efficient to run it on the latter: the attack of [21] takes only a bit more than four days to run on a single GTX 970, which is much less than the estimated 150 days it would take using a single quad-core CPU.

We did not write a CPU (resp. GPU) implementation of our own attack for the search of the second (resp. first) block, and are thus unable to make a similar comparison for the present full hash function attack. However, as we used the same framework as [21], it is reasonable to assume that the gap would be of the same order.

How to pick the best generic attack. As we pointed out above, the common methodology for measuring the complexity of an attack leaves implicit the comparison with a generic approach. This may introduce a bias in suggesting a strategy for a generic attacker that is in fact not optimal. This was already hinted in the previous paragraph, where we remarked that an attack may seem to become worse when implemented on a more efficient platform. In fact, the underlying assumption that a generic attacker would use the same platform as the one on which the cryptanalytic attack is implemented may not always be justified: for instance, even if the latter is run on a CPU, there is no particular reason why a generic attacker would not use more energy-efficient GPUs or FPGAs. It may thus be hard to precisely estimate the absolute gain provided by a cryptanalytic attack compared to the best implementation of a generic algorithm *with identical monetary and time resources*, especially when these are high.

The issues raised here could all be addressed in principle by carefully implementing, say van Oorschot and Wiener's parallel collision search on a cluster of efficient platforms [33]. However, this is usually not done in practice, and we made no exception in our case.

Despite the few shortcomings of this usual methodology used to evaluate the complexity of attacks, it remains in our opinion a reliable measure thereof, that allows to compare different attack efforts reasonably well. For want of a better one, it is also the approach used in this paper.

6.2 The computation

The major challenge when running our near-collision attacks distributed across the world was to adapt it into a distributed computation model which pursues two goals: the geographically distributed workers should work independently without duplication of work, and the number of the wasted computational time due to worker's failures should be minimized. The first goal required storage with the ability endure high loads of requests coming from all around the globe. For the second goal, the main sources of failures we found were preemption by higher-priority workers and bugs in GPU hardware. To diminish the impact of these failures, we learned to predict failures in the early stages of computation and terminated workers without wasting significant amounts of computational time.

First near-collision attack. The first phase of the attack, corresponding to the generation of first-block near collisions, was run on a heterogeneous CPU cluster hosted by Google, spread over eight physical locations. The computation was split into small jobs of expected running time of one hour, whose objectives were to compute partial solutions up to step 61. The running time of one hour proved to be the best choice to be resilient against various kind of failures (mostly machine failure, preemption by other users of the cluster, or network issues), while limiting the overhead of managing many jobs. A *MapReduce* paradigm was used to collect the solutions of a series of smaller jobs; in hindsight, this was not the best approach, as it introduced an unnecessary bottleneck in the reduce phase.

The first first-block near collision was found after spending about 3583 core years that had produced 180 711 partial solutions up to step 61. A second near collision block was then later computed; it required an additional 2987 core years and 148 975 partial solutions.

There was a variety of CPUs involved in this computation, but it is reasonable to assume that they all were roughly equivalent in performance. On a single core of a 2.3 GHz Xeon E5-2650v3, the OpenSSL implementation of SHA-1 can compute up to $2^{23.3}$ compression functions per second. Taking this as a unit, the first near-collision block required an effort equivalent to 2^{60} SHA-1 compression function calls, and the second first block required $2^{59.75}$.

Second near-collision attack. The second more expensive phase of the attack was run on a heterogeneous cluster of K20, K40 and K80 GPUs, also hosted by Google. It corresponded to the generation of a second-block near-collision leading to a full collision.

The overall setup of the computation was similar to the one of the first block, except that it did not use a MapReduce approach and resorted to using simpler queues holding the unprocessed jobs. A worker would then select a job, potentially produce one or several partial solutions up to step 61, and die on completion.

The collision was found after 369 985 partial solutions had been produced[5]. The production rates of partial 61-step solutions of the different devices used in the cluster were of 0.593 per hour for the K80 (which combines two GPU chips on one card), 0.444 for the K40 and 0.368 for the K20. The time needed for a homogeneous cluster to produce the collision would then have been of 114 K20-years, 95 K40-years or 71 K80-years.

The rate at which these various devices can compute the compression function of SHA-1 is, according to our measurements, $2^{31.1}\,s^{-1}$ for the K20, $2^{31.3}\,s^{-1}$ for the K40, and $2^{31}\,s^{-1}$ for the K80 ($2^{30}\,s^{-1}$ per GPU). The effort of finding the second block of the collision for homogeneous clusters, measured in number of equivalent calls to the compression function, is thus equal to $2^{62.8}$ for the K20 and K40 and $2^{62.1}$ for the K80.

Although a GTX 970 was only used to prototype the attack, we can also consider its projected efficiency and measure the effort spent for the attack w.r.t. this GPU. From the measured production rate of 58 100 step 33 solutions per second, we can deduce that 0.415 step 61 solutions can be computed per hour on average. This leads to a computational effort of 102 GPU years, equivalent to $2^{63.4}$ SHA-1 compression function calls.

The monetary cost of computing the second block of the attack by renting Amazon instances can be estimated from these various data. Using a `p2.16xlarge` instance, featuring 16 K80 GPUs and nominally costing US$ 14.4 per hour would cost US$ 560 K for the necessary 71 device years. It would be more economical for a patient attacker to wait for low "spot prices" of the smaller `g2.8xlarge` instances, which feature four K520 GPUs, roughly equivalent to a K40 or a GTX 970. Assuming thusly an effort of 100 device years, and a typical spot price of US$ 0.5 per hour, the overall cost would be of US$ 110 K.

Finally, summing the cost of each phase of the attack in terms of compression function calls, we obtain a total effort of $2^{63.1}$, including the redundant second near-colliding first block and taking the figure of $2^{62.8}$ for the second block collision. This should however not be taken as an absolute number; depending on luck and equipment but without changing any of the cryptanalytical aspects of our attack, it is conceivable that the

[5] We were quite lucky in that respect. The expected number required is about 2.5 times more than that.

spent effort could have been anywhere from, say, $2^{62.3}$ to $2^{65.1}$ equivalent compression function calls.

6.3 Complexity comparisons

We put our own result into perspective by briefly comparing its complexity to a few other relevant cryptographic computations.

Comparison with MD5 and SHA-0 collisions. An apt comparison is first to consider the cost of computing collisions for MD5 [37], a once very popular hash function, and SHA-0 [30], identical to SHA-1 but for a missing rotation in the message expansion. The most efficient known identical-prefix collision attacks for these three functions are all based on the same series of work from Wang *et al.* from the mid-2000s [49, 50, 48], but have widely varying complexities.

The best current identical-prefix collision attacks on MD5 are due to Stevens *et al.*, and require the equivalent of about 2^{16} compression function calls [46]. Furthermore, in the same paper, *chosen-prefix* collisions are computed for a cost equivalent to about 2^{39} calls, increasing to 2^{49} calls for a three-block chosen-prefix collision as was generated on 200 PS3s for the rogue Certification Authority work.

Though very similar to SHA-1, SHA-0 is much weaker against collision attacks. The best current such attack on SHA-0 is due to Manuel and Peyrin [27], and requires the equivalent of about $2^{33.6}$ calls to the compression function.

Identical-prefix collisions for MD5 and SHA-0 can thus be obtained within a reasonable time by using very limited computational power, such as a decent smartphone.

Comparison with RSA modulus factorization and prime field discrete logarithm computation. Some of the most expensive attacks implemented in cryptography are in fact concerned with establishing records of factorization and discrete logarithm computations. We believe that it is instructive to compare the resources necessary in both cases. As an example, we consider the 2009 factorization of a 768-bit RSA modulus from Kleinjung *et al.* [22] and the recent 2016 discrete logarithm computation in a 768-bit prime field from Kleinjung *et al.* [23].

The 2009 factorization required about 2000 core years on a 2.2 GHz AMD Opteron of the time. The number of single instructions to have been executed is estimated to be of the order of 2^{67} [22][6].

[6] Note that the comparison between factorization and discrete logarithm computation mentioned in [23] gives for the former a slightly lower figure of about 1700 core years.

The 2016 discrete logarithm computation was a bit more than three times more expensive, and required about 5300 core years on a single core of a 2.2 GHz Xeon E5-2660 [23].

In both cases, the overall computational effort could have been decreased by reducing the time that was spent collecting relations [22, 23]. However, this would have made the following linear-algebra step harder to manage and a longer computation in calendar time. Kleinjung *et al.* estimated that a shorter sieving step could have resulted in a discrete logarithm computation in less than 4000 core years [23].

To compare the cost of the attacks, we can estimate how many SHA-1 (compression function) calls can be performed in the 5300 core years of the more expensive discrete logarithm record [23]. Considering again a 2.3 GHz Xeon E5-2650 (slightly faster than the CPU used as a unit by Kleinjung *et al.*) running at about $2^{23.3}$ SHA-1 calls per second, the overall effort of [23] is equivalent to approximately $2^{60.6}$ SHA-1 calls. It is reasonable to expect that even on an older processor the performance of running SHA-1 would not decrease significantly; taking the same base figure per core would mean that the effort of [22] is equivalent to approximately $2^{58.9} \sim 2^{59.2}$ SHA-1 calls.

In absolute value, this is less than the effort of our own attack, the more expensive discrete logarithm computation being about five times cheaper[7], and less than twice more expensive than computing a single first-block near collision. However, the use of GPUs for the computation of the second block of our attack allowed both to significantly decrease the calendar time necessary to perform the computation, and its efficiency in terms of necessary power: as an example, the peak power consumption of a K40 is only 2.5 times the one of a 10-core Xeon E5-2650, yet it is about 25 times faster at computing the compression function of SHA-1 than the whole CPU, and thence 10 times more energy-efficient overall. The energy required to compute a collision using GPUs is thus about twice less than the one required for the discrete logarithm computation[8]. As a conclusion, computing a collision for SHA-1 seems to need slightly more effort than 768-bit RSA factorization or prime-field discrete logarithm computation but, if done on GPUs, the amount of resources necessary to do so is slightly less.

[7] But now is also a good time to recall that directly comparing CPU and GPU cost is tricky.

[8] This is assuming that the total energy requirements scale linearly with the consumption of the processing units.

Acknowledgements. We thank the anonymous reviewers for their helpful comments, and Michael X. Lyons for pointing out a few minor inconsistencies between the presented differential path and the actual colliding blocks.

References

[1] Albertini, A., Aumasson, J., Eichlseder, M., Mendel, F., Schläffer, M.: Malicious Hashing: Eve's Variant of SHA-1. In: Joux, A., Youssef, A.M. (eds.) SAC 2014. Lecture Notes in Computer Science, vol. 8781, pp. 1–19. Springer (2014)
[2] Albertini, A., *al.*: Exploiting identical-prefix hash function collisions. Draft (2017)
[3] Biham, E., Chen, R.: Near-Collisions of SHA-0. In: Franklin, M.K. (ed.) CRYPTO. Lecture Notes in Computer Science, vol. 3152, pp. 290–305. Springer (2004)
[4] Biham, E., Chen, R., Joux, A., Carribault, P., Lemuet, C., Jalby, W.: Collisions of SHA-0 and Reduced SHA-1. In: Cramer [6], pp. 36–57
[5] Chabaud, F., Joux, A.: Differential Collisions in SHA-0. In: Krawczyk, H. (ed.) CRYPTO. Lecture Notes in Computer Science, vol. 1462, pp. 56–71. Springer (1998)
[6] Cramer, R. (ed.): EUROCRYPT, Lecture Notes in Computer Science, vol. 3494. Springer (2005)
[7] De Cannière, C., Mendel, F., Rechberger, C.: Collisions for 70-Step SHA-1: On the Full Cost of Collision Search. In: Adams, C.M., Miri, A., Wiener, M.J. (eds.) SAC. Lecture Notes in Computer Science, vol. 4876, pp. 56–73. Springer (2007)
[8] De Cannière, C., Rechberger, C.: Finding SHA-1 Characteristics: General Results and Applications. In: Lai, X., Chen, K. (eds.) ASIACRYPT. Lecture Notes in Computer Science, vol. 4284, pp. 1–20. Springer (2006)
[9] den Boer, B., Bosselaers, A.: An Attack on the Last Two Rounds of MD4. In: Feigenbaum, J. (ed.) CRYPTO. Lecture Notes in Computer Science, vol. 576, pp. 194–203. Springer (1991)
[10] den Boer, B., Bosselaers, A.: Collisions for the Compression Function of MD5. In: Helleseth, T. (ed.) EUROCRYPT. Lecture Notes in Computer Science, vol. 765, pp. 293–304. Springer (1993)
[11] Dobbertin, H.: Cryptanalysis of MD4. In: Gollmann, D. (ed.) FSE. Lecture Notes in Computer Science, vol. 1039, pp. 53–69. Springer (1996)
[12] Fillinger, M., Stevens, M.: Reverse-Engineering of the Cryptanalytic Attack Used in the Flame Super-Malware. In: Iwata, T., Cheon, J.H. (eds.) ASIACRYPT. Lecture Notes in Computer Science, vol. 9453, pp. 586–611. Springer (2015)
[13] Forum, C.: Ballot 152 - Issuance of SHA-1 certificates through 2016. Cabforum mailing list (2015), https://cabforum.org/pipermail/public/2015-October/006081.html
[14] Gebhardt, M., Illies, G., Schindler, W.: A note on practical value of single hash collisions for special file formats. NIST First Cryptographic Hash Workshop (Oct 2005)
[15] Grechnikov, E.: Collisions for 72-step and 73-step sha-1: Improvements in the method of characteristics. Cryptology ePrint Archive, Report 2010/413 (2010)
[16] Grechnikov, E., Adinetz, A.: Collision for 75-step sha-1: Intensive parallelization with gpu. Cryptology ePrint Archive, Report 2011/641 (2011)
[17] Hashclash project webpage (Retrieved May 2017), https://marc-stevens.nl/p/hashclash/

[18] InfoWorld: Oracle to java devs: Stop signing jar files with md5 (January 2017)
[19] Joux, A., Peyrin, T.: Hash Functions and the (Amplified) Boomerang Attack. In: Menezes, A. (ed.) CRYPTO. Lecture Notes in Computer Science, vol. 4622, pp. 244–263. Springer (2007)
[20] Jutla, C.S., Patthak, A.C.: A Matching Lower Bound on the Minimum Weight of SHA-1 Expansion Code. IACR Cryptology ePrint Archive 2005, 266 (2005)
[21] Karpman, P., Peyrin, T., Stevens, M.: Practical free-start collision attacks on 76-step SHA-1. In: Gennaro, R., Robshaw, M. (eds.) CRYPTO. Lecture Notes in Computer Science, vol. 9215, pp. 623–642. Springer (2015)
[22] Kleinjung, T., Aoki, K., Franke, J., Lenstra, A.K., Thomé, E., Bos, J.W., Gaudry, P., Kruppa, A., Montgomery, P.L., Osvik, D.A., te Riele, H.J.J., Timofeev, A., Zimmermann, P.: Factorization of a 768-Bit RSA Modulus. In: Rabin, T. (ed.) CRYPTO 2010. Lecture Notes in Computer Science, vol. 6223, pp. 333–350. Springer (2010)
[23] Kleinjung, T., Diem, C., Lenstra, A.K., Priplata, C., Stahlke, C.: Computation of a 768-Bit Prime Field Discrete Logarithm. In: Coron, J., Nielsen, J.B. (eds.) EUROCRYPT. Lecture Notes in Computer Science, vol. 10210, pp. 185–201 (2017)
[24] Lab, C.: skywiper (a.k.a. flame a.k.a. flamer): A complex malware for targeted attacks. Laboratory of Cryptography and System Security, Budapest University of Technology and Economics (May 31, 2012)
[25] Lab, K.: The flame: Questions and answers. Securelist blog (May 28, 2012)
[26] Manuel, S.: Classification and generation of disturbance vectors for collision attacks against SHA-1. Des. Codes Cryptography 59(1-3), 247–263 (2011)
[27] Manuel, S., Peyrin, T.: Collisions on SHA-0 in One Hour. In: Nyberg, K. (ed.) FSE. Lecture Notes in Computer Science, vol. 5086, pp. 16–35. Springer (2008)
[28] Mendel, F., Pramstaller, N., Rechberger, C., Rijmen, V.: The impact of carries on the complexity of collision attacks on SHA-1. In: Robshaw, M.J.B. (ed.) FSE. Lecture Notes in Computer Science, vol. 4047, pp. 278–292. Springer (2006)
[29] third author's mum, T.: Sha-1 is still being used. Personnal communication
[30] National Institute of Standards and Technology: FIPS 180: Secure Hash Standard (May 1993)
[31] National Institute of Standards and Technology: FIPS 180-1: Secure Hash Standard (April 1995)
[32] Nossum, V.: SAT-based preimage attacks on SHA-1. Master's thesis, University of Oslo (2012)
[33] van Oorschot, P.C., Wiener, M.J.: Parallel Collision Search with Cryptanalytic Applications. J. Cryptology 12(1), 1–28 (1999)
[34] Post, T.W.: U.s., israel developed flame computer virus to slow iranian nuclear efforts, officials say (June 2012)
[35] Pramstaller, N., Rechberger, C., Rijmen, V.: Exploiting coding theory for collision attacks on SHA-1. In: Smart, N.P. (ed.) Cryptography and Coding, 10th IMA International Conference. Lecture Notes in Computer Science, vol. 3796, pp. 78–95. Springer (2005)
[36] Rivest, R.L.: The MD4 message digest algorithm. In: Menezes, A., Vanstone, S.A. (eds.) CRYPTO. Lecture Notes in Computer Science, vol. 537, pp. 303–311. Springer (1990)
[37] Rivest, R.L.: RFC 1321: The MD5 Message-Digest Algorithm (April 1992)
[38] Schneier, B.: When will we see collisions for sha-1? Blog (2012)
[39] Services, A.W.: Amazon EC2 – Virtual Server Hosting. aws.amazon.com (Retrieved Jan 2016)

[40] Shoup, V. (ed.): CRYPTO, Lecture Notes in Computer Science, vol. 3621. Springer (2005)
[41] Stevens, M.: Attacks on Hash Functions and Applications. Ph.D. thesis, Leiden University (June 2012)
[42] Stevens, M.: Counter-Cryptanalysis. In: Canetti, R., Garay, J.A. (eds.) CRYPTO. Lecture Notes in Computer Science, vol. 8042, pp. 129–146. Springer (2013)
[43] Stevens, M.: New Collision Attacks on SHA-1 Based on Optimal Joint Local-Collision Analysis. In: Johansson, T., Nguyen, P.Q. (eds.) EUROCRYPT. Lecture Notes in Computer Science, vol. 7881, pp. 245–261. Springer (2013)
[44] Stevens, M., Karpman, P., Peyrin, T.: Freestart collision for full SHA-1. In: Fischlin, M., Coron, J.S. (eds.) EUROCRYPT. Lecture Notes in Computer Science, vol. 9665, pp. 459–483. Springer (2016)
[45] Stevens, M., Lenstra, A.K., de Weger, B.: Chosen-Prefix Collisions for MD5 and Colliding X.509 Certificates for Different Identities. In: Naor, M. (ed.) EUROCRYPT. Lecture Notes in Computer Science, vol. 4515, pp. 1–22. Springer (2007)
[46] Stevens, M., Sotirov, A., Appelbaum, J., Lenstra, A.K., Molnar, D., Osvik, D.A., de Weger, B.: Short Chosen-Prefix Collisions for MD5 and the Creation of a Rogue CA Certificate. In: Halevi, S. (ed.) CRYPTO. Lecture Notes in Computer Science, vol. 5677, pp. 55–69. Springer (2009)
[47] ThreadPost: Sha-1 end times have arrived (January 2017)
[48] Wang, X., Yin, Y.L., Yu, H.: Finding Collisions in the Full SHA-1. In: Shoup [40], pp. 17–36
[49] Wang, X., Yu, H.: How to Break MD5 and Other Hash Functions. In: Cramer [6], pp. 19–35
[50] Wang, X., Yu, H., Yin, Y.L.: Efficient collision search attacks on SHA-0. In: Shoup [40], pp. 1–16
[51] Yajima, J., Iwasaki, T., Naito, Y., Sasaki, Y., Shimoyama, T., Peyrin, T., Kunihiro, N., Ohta, K.: A strict evaluation on the number of conditions for SHA-1 collision search. IEICE Transactions 92-A(1), 87–95 (2009), http://search.ieice.org/bin/summary.php?id=e92-a_1_87&category=A&year=2009&lang=E&abst=
[52] Yajima, J., Sasaki, Y., Naito, Y., Iwasaki, T., Shimoyama, T., Kunihiro, N., Ohta, K.: A new strategy for finding a differential path of SHA-1. In: Pieprzyk, J., Ghodosi, H., Dawson, E. (eds.) ACISP. Lecture Notes in Computer Science, vol. 4586, pp. 45–58. Springer (2007)

Table 3: Meaning of the bit difference symbols, for a symbol located on $A_t[i]$. The same symbols are also used for m.

Symbol	Condition on $(A,\widetilde{A})$	Symbol	Condition on $(A,\widetilde{A})$
$\cdot$	$A_t[i] = \widetilde{A}_t[i]$	$\star$	$A_t[i] = \widetilde{A}_t[i] = A_{t-1}[i]$
$\bullet$	$A_t[i] \neq \widetilde{A}_t[i]$	$\star$	$A_t[i] = \widetilde{A}_t[i] \neq A_{t-1}[i]$
$\blacktriangle$	$A_t[i] = 0, \quad \widetilde{A}_t[i] = 1$	$\diamond$	$A_t[i] = \widetilde{A}_t[i] = (A_{t-1}^{\circlearrowright 2})[i]$
$\blacktriangledown$	$A_t[i] = 1, \quad \widetilde{A}_t[i] = 0$	$\blacklozenge$	$A_t[i] = \widetilde{A}_t[i] \neq (A_{t-1}^{\circlearrowright 2})[i]$
$\triangledown$	$A_t[i] = \widetilde{A}_t[i] = 0$	$\square$	$A_t[i] = \widetilde{A}_t[i] = (A_{t-2}^{\circlearrowright 2})[i]$
$\triangle$	$A_t[i] = \widetilde{A}_t[i] = 1$	$\blacksquare$	$A_t[i] = \widetilde{A}_t[i] \neq (A_{t-2}^{\circlearrowright 2})[i]$
$*$	No condition on $A_t[i]$, $\widetilde{A}_t[i]$		

Table 4: Additional necessary conditions used for A_{22} to A_{26}.

$$
\begin{aligned}
&A_{22}[27] \oplus m_{23}[27] = A_{21}[29] \oplus 1\\
&A_{24}[27] \oplus m_{25}[27] = A_{23}[29]\\
&A_{25}[28] \oplus m_{25}[27] = A_{23}[30] \oplus 1\\
&A_{26}[27] \oplus m_{27}[27] = A_{25}[29]\\
&\begin{cases} A_{25}[29] \oplus m_{23}[27] = A_{24}[31] & \text{if } A_{24}[30] = m_{23}[30]\\ A_{24}[31] = m_{23}[30] & \text{if } A_{24}[30] \neq m_{23}[30] \end{cases}
\end{aligned}
$$

A The attack parameters

The first block of the attack uses the same path and conditions as the one given in [43, Section 5], which we refer to for a description. This section gives the differential path, linear (message) bit-relations and neutral bits used in our second-block near-collision attack.

We use the notation of Table 3 to represent signed differences of the differential path and to indicate the position of neutral bits.

We give the differential path of the second block up to A_{23} in Figure 3. We also give *necessary conditions* for A_{22} to A_{26} in Table 4, which are required for all alternate differential paths allowed. In order to maximize the probability, some additional conditions are also imposed on the message. These message-bit-relations are given in Table 5. The rest of the path can then be determined from the disturbance vector.

We also give the list of the neutral bits used in the attack. There are 51 of them over the seven message words m_{11} to m_{15}, distributed as follows (visualized in Figure 4):

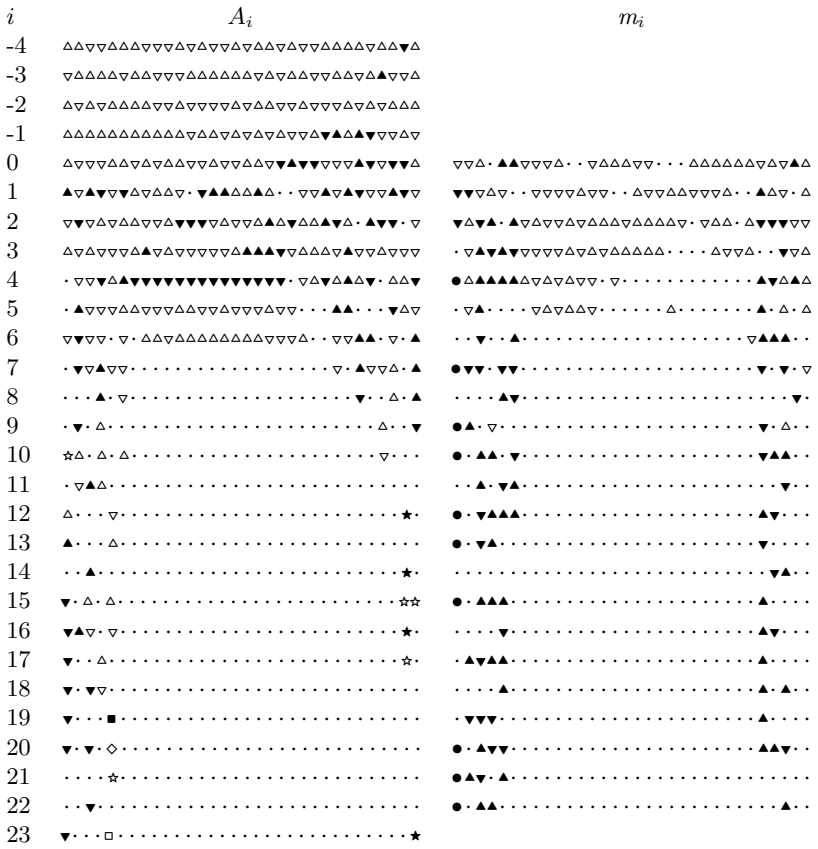

Fig. 3: The differential path of the second block up to A_{23}.

Table 5: Linear part message-bit-relations for the second block path.

$m_{23}[27] \oplus m_{23}[28] = 1$	$m_{23}[30] \oplus m_{24}[3] = 1$	$m_{23}[30] \oplus m_{28}[28] = 1$
$m_{23}[4] = 0$	$m_{24}[28] = 0$	$m_{24}[29] = 0$
$m_{24}[2] = 0$	$m_{26}[28] \oplus m_{26}[29] = 1$	$m_{27}[29] = 0$
$m_{28}[27] = 0$	$m_{28}[4] \oplus m_{32}[29] = 0$	$m_{36}[4] \oplus m_{44}[28] = 1$
$m_{38}[4] \oplus m_{44}[28] = 0$	$m_{39}[30] \oplus m_{44}[28] = 1$	$m_{40}[3] \oplus m_{44}[28] = 0$
$m_{40}[4] \oplus m_{44}[28] = 1$	$m_{41}[29] \oplus m_{41}[30] = 0$	$m_{42}[28] \oplus m_{44}[28] = 0$
$m_{43}[28] \oplus m_{44}[28] = 0$	$m_{43}[29] \oplus m_{44}[28] = 1$	$m_{43}[4] \oplus m_{47}[29] = 0$
$m_{44}[28] \oplus m_{44}[29] = 1$	$m_{45}[29] \oplus m_{47}[29] = 0$	$m_{46}[29] \oplus m_{47}[29] = 0$
$m_{48}[4] \oplus m_{52}[29] = 0$	$m_{50}[29] \oplus m_{52}[29] = 0$	$m_{51}[29] \oplus m_{52}[29] = 0$
$m_{54}[4] \oplus m_{60}[29] = 1$	$m_{56}[29] \oplus m_{60}[29] = 1$	$m_{56}[4] \oplus m_{60}[29] = 0$
$m_{57}[29] \oplus m_{60}[29] = 1$	$m_{59}[29] \oplus m_{60}[29] = 0$	$m_{67}[0] \oplus m_{72}[30] = 1$
$m_{68}[5] \oplus m_{72}[30] = 0$	$m_{70}[1] \oplus m_{71}[6] = 1$	$m_{71}[0] \oplus m_{76}[30] = 1$
$m_{72}[5] \oplus m_{76}[30] = 0$	$m_{73}[2] \oplus m_{78}[0] = 1$	$m_{74}[1] \oplus m_{75}[6] = 1$
$m_{74}[7] \oplus m_{78}[0] = 0$	$m_{75}[1] \oplus m_{76}[6] = 1$	$m_{76}[0] \oplus m_{76}[1] = 1$
$m_{76}[3] = 1$	$m_{77}[0] \oplus m_{77}[1] = 0$	$m_{77}[0] \oplus m_{77}[2] = 1$
$m_{77}[8] = 0$	$m_{78}[3] = 1$	$m_{78}[7] = 0$
$m_{79}[2] = 0$	$m_{79}[4] = 1$	

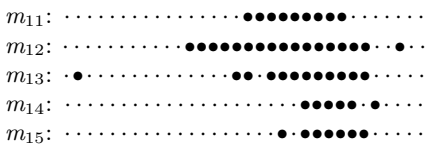

Fig. 4: The 51 single neutral bits used in the second block attack.

- m_{11}: bit positions (starting with the least significant bit at zero) 7, 8, 9, 10, 11, 12, 13, 14, 15
- m_{12}: positions 2, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20
- m_{13}: positions 5, 6, 7, 8, 9, 10, 11, 12, 13, 15, 16, 30
- m_{14}: positions 4, 6, 7, 8, 9, 10
- m_{15}: positions 5, 6, 7, 8, 9, 10, 12

Not all of the neutral bits of the same word (say m_{13}) are neutral up to the same point. Their repartition in that respect is as follows, a graphical representation being also given in Figure 5.

- Bits neutral up to A_{14} (included): m_{11}[9,10,11,12,13,14,15], m_{12}[2,14,15,16,17,18,19,20], m_{13}[12,16]
- Bits neutral up to A_{15} (included): m_{11}[7,8], m_{12}[9,10,11,12,13], m_{13}[15,30]
- Bits neutral up to A_{16} (included): m_{12}[5,6,7,8], m_{13}[10,11,13]
- Bits neutral up to A_{17} (included): m_{13}[5,6,7,8,9], m_{14}[10]

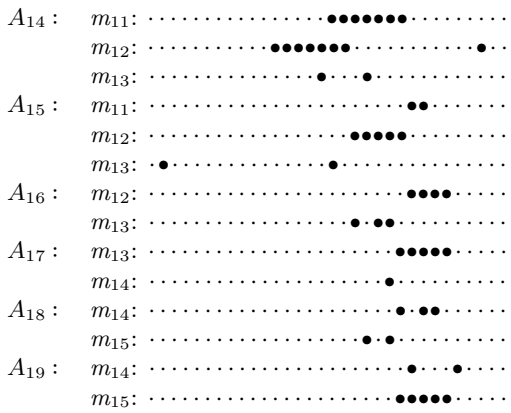

Fig. 5: The 51 single neutral bits regrouped by up to where they are neutral.

– Bits neutral up to A_{18} (included): m_{14}[6,7,9], m_{15}[10,12]
– Bits neutral up to A_{19} (included): m_{14}[4,8], m_{15}[5,6,7,8,9]

A bit neutral to A_i is then used to produce partial solutions at A_{i+1}. One should also note that this list only includes a single bit per neutral bit group, and some additional flips may be necessary to preserve message-bit-relations.

Out of the three boomerangs used in the attack, one first introduced a perturbation on m_9 on bit 7, and the other two on m_6, on bit 6 and on bit 8. All three boomerangs then introduce corrections to ensure a local collision. Because these local collisions happen in the first round, where the Boolean function is φ_{IF}, only two corrections are necessary for each of them.

The lone boomerang introduced on m_9 is neutral up to A_{22}, and the couple introduced on m_6 are neutral up to A_{25}. The complete sets of message bits defining all of them are shown in Figure 6, using a "difference notation".

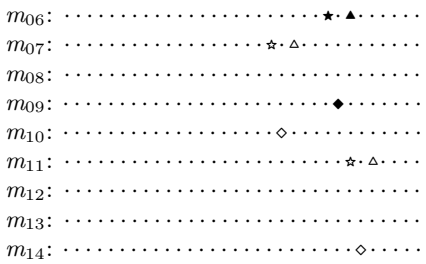

Fig. 6: Boomerang local collision patterns using symbols. The first perturbation difference is highlighted with a black symbol. Associated correcting differences are identified with the corresponding white symbol.