

Freestart collision for full SHA-1

Marc Stevens¹, Pierre Karpman^{2,3,4}, and Thomas Peyrin⁴

¹ Centrum Wiskunde & Informatica, The Netherlands

² Inria, France


³ École polytechnique, France


⁴ Nanyang Technological University, Singapore


marc.stevens@cwi.nl, pierre.karpman@inria.fr, thomas.peyrin@ntu.edu.sg

Abstract. This article presents an explicit freestart colliding pair for **SHA-1**, *i.e.* a collision for its internal compression function. This is the first practical break of the full **SHA-1**, reaching all 80 out of 80 steps. Only 10 days of computation on a 64-GPU cluster were necessary to perform this attack, for a runtime cost equivalent to approximately $2^{57.5}$ calls to the compression function of **SHA-1** on GPU. This work builds on a continuous series of cryptanalytic advancements on **SHA-1** since the theoretical collision attack breakthrough of 2005. In particular, we reuse the recent work on 76-step **SHA-1** of Karpman *et al.* from CRYPTO 2015 that introduced an efficient framework to implement (freestart) collisions on GPUs; we extend it by incorporating more sophisticated accelerating techniques such as boomerangs. We also rely on the results of Stevens from EUROCRYPT 2013 to obtain optimal attack conditions; using these techniques required further refinements for this work. Freestart collisions do not directly imply a collision for the full hash function. However, this work is an important milestone towards an actual **SHA-1** collision and it further shows how GPUs can be used very efficiently for this kind of attack. Based on the state-of-the-art collision attack on **SHA-1** by Stevens from EUROCRYPT 2013, we are able to present new projections on the computational and financial cost required for a **SHA-1** collision computation. These projections are significantly lower than what was previously anticipated by the industry, due to the use of the more cost efficient GPUs compared to regular CPUs. We therefore recommend the industry, in particular Internet browser vendors and Certification Authorities, to retract **SHA-1** quickly. We hope the industry has learned from the events surrounding the cryptanalytic breaks of MD5 and will retract **SHA-1** before concrete attacks such as signature forgeries appear in the near future.

Keywords: **SHA-1**, hash function, cryptanalysis, freestart collision, GPU implementation.

 Supported by the Netherlands Organization for Scientific Research Veni Grant 2014

 Partially supported by the Direction Générale de l'Armement and by the Singapore National Research Foundation Fellowship 2012 (NRF-NRFF2012-06)

 Supported by the Singapore National Research Foundation Fellowship 2012 (NRF-NRFF2012-06)

1 Introduction

A cryptographic hash function H is a function that takes an arbitrarily long message M as input and outputs a fixed-length hash value of size n bits. It is a versatile primitive useful in many applications, such as building digital signature schemes, message authentication codes or password hashing functions. One key security feature expected from a cryptographic hash function is collision resistance: it should not be feasible for an adversary to find two distinct messages M, \hat{M} that hash to the same value $H(M) = H(\hat{M})$ faster than with a generic algorithm, *i.e.* with significantly less than $2^{\frac{n}{2}}$ calls to the hash function.

A widely used hash function construction is the Merkle-Damgård paradigm [22,4]: H is built by iterating a compression function h that updates a fixed-size internal state (also called chaining value) with fixed-size message blocks; the initial chaining value (IV) is a fixed constant of the hash function. This construction is useful in particular for the simple security reduction it allows to make: if the compression function is collision-resistant, then so is the corresponding hash function. This leads to defining variants of collision attacks which allow the attacker to choose the IV: a *freestart* collision is a pair of different message and IV $(C, M), (\hat{C}, \hat{M})$ such that $H_C(M) = H_{\hat{C}}(\hat{M})$; *semi-freestart* collisions are similar but impose $C = \hat{C}$. It is noteworthy that the Merkle-Damgård security reduction assumes that any type of collision (freestart or semi-freestart) must be intractable by the adversary. Thus, a collision attack on the compression function should be taken very seriously as it invalidates the security reduction coming from the operating mode of the hash function.

The most famous hash function family, basis for most hash function industry standards, is undoubtedly the MD-SHA family, which includes notable functions such as MD4, MD5, SHA-1 and SHA-2. This family first originated with MD4 [33] and continued with MD5 [34] (due to serious security weaknesses [7,9] found on MD4 soon after its publication). Even though collision attacks on the compression function were quickly identified [8], the industry widely deployed MD5 in applications where hash functions were required. Yet, in 2005, a team of researchers led by Wang [44] completely broke the collision resistance of MD5, which allowed to efficiently compute colliding messages for the full hash function. This groundbreaking work inspired much further research on the topic; in a major development, Stevens *et al.* [41] 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 [42]. This past history of cryptanalysis on MD5 is yet another argument for a very careful treatment of collision cryptanalysis progress: the industry should move away from weak cryptographic hash functions or hash functions built on weak inner components (compression functions that are not collision resistant) before the seemingly theoretic attacks prove to be a direct threat to security (*counter-cryptanalysis* [39] could be used to mitigate some of the risks during the migration).

While lessons should be learned from the case of MD5, it is interesting to observe that the industry is again facing a similar challenge. SHA-1 [26], designed by the NSA and a NIST standard, is one of the main hash functions of today, and it is facing important attacks since 2005. Based on previous successful cryptanalysis works [3,1,2] on SHA-0 [25] (SHA-1's predecessor, that only differs by a single rotation in the message expansion function), a team led again by Wang *et al.* [43] showed in 2005 the very first theoretical collision attack on SHA-1. Unlike the case of MD5, this attack, while groundbreaking, remains mostly theoretical as its expected cost was evaluated to be equivalent to 2^{69} calls to the SHA-1 compression function.

Therefore, as a proof of concept, many teams considered generating real collisions for reduced versions of SHA-1: 64 steps [6] (with a cost of 2^{35} SHA-1 calls), 70 steps [5] (cost 2^{44} SHA-1), 73 steps [12] (cost $2^{50.7}$ SHA-1) and the latest advances for the hash function reached 75 steps [13] (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 [40] with an estimated cost of 2^{61} calls to the SHA-1 compression function. Nevertheless, a publicly known collision still remains out of reach.

Very recently, collisions on the compression function of SHA-1 reduced to 76 steps (out of 80) were obtained by using a start-from-the-middle approach and a highly efficient GPU framework [17]. This required only a reasonable amount of GPU computation power (less than a week on a single card, equivalent to about $2^{50.3}$ calls to SHA-1 on GPU, whereas the runtime cost equivalent on regular CPUs is about $2^{49.1}$ SHA-1).

Because of these worrisome cryptanalysis advances on SHA-1, one is advised to use *e.g.* SHA-2 [27] or the new hash functions standard SHA-3 [29] when secure hashing is needed. While NIST recommended that SHA-1-based certificates should not be trusted beyond 2014 [28] (by 2010 for governmental use), the industry actors only recently started to

move away from **SHA-1**, about a decade after the first theoretical collision attacks. For example, Microsoft, Google and Mozilla have all announced that their respective browsers will stop accepting **SHA-1** SSL certificates by 2017 (and that **SHA-1**-based certificates should not be issued after 2015). These deadlines are motivated by a simple evaluation of the computational and financial cost required to generate a collision for **SHA-1**: in 2012, Bruce Schneier (using calculations by Jesse Walker based on a 2^{61} attack cost [40], Amazon EC2 spotprices and Moore’s Law) estimated the cost of running one **SHA-1** collision attack to be around 700,000 US\$ in 2015, down to about 173,000 US\$ in 2018, which he deemed to be within the resources of criminals [35]. We observe that while a majority of industry actors already chose to migrate to more secure hashing algorithms, surveys show that in September 2015 **SHA-1** remained the hashing primitive for about 28.2% of certificate signatures [37].

1.1 Our contributions

In this article, we give the first colliding pair for the full **SHA-1** compression function (see Table 1-1), which amounts to a freestart collision for the full hash function. This was obtained at a GPU runtime cost approximately equivalent to $2^{57.5}$ evaluations of **SHA-1**.¹

The starting point for this attack is the start-from-the-middle approach and the GPU framework of CRYPTO 2015, which was used to compute freestart collisions on the 76-step reduced **SHA-1** [17]. We improve this by incorporating the auxiliary paths (or boomerangs) speed-up technique from Joux and Peyrin [15]. We also rely on the cryptanalytic techniques by Stevens [40] to obtain optimal attack conditions, which required further refinements for this work.

As was mentioned above, previous recommendations on retracting **SHA-1** were based on estimations of the resources needed to find **SHA-1** collisions. These consist both in the time necessary to mount an attack, for a given computational power, as well as the cost of building and maintaining this capability, or of renting the equipment directly on a platform such as Amazon EC2 [36]. In that respect, our freestart collision attack can be run in about 9 to 10 days on average on a cluster with 64 GeForce GTX970 GPUs, or by renting GPU time on Amazon EC2 for about 2K US\$.² Based on this experimental data and the 2013 state-of-

¹ Which from previous experience is about a factor 2 higher than the runtime cost in equivalent number of **SHA-1** evaluations on regular CPUs.

² This is based on the spot price for Amazon EC2 GPU Instance Type ‘g2.8xlarge’, featuring 4 GPUs, which is about 0.50 US\$ per hour as of October 2015. These four

Table 1-1. A freestart collision for SHA-1. A test program for this colliding pair is available at <https://sites.google.com/site/itstheshappening/tester.cpp>

Message 1																																	
IV_1	50	6b	01	78	ff	6d	18	90 20	22	91	fd	3a	de	38	71	b2	c6	65	ea														
M_1	9d	44	38	28 a5	ea	3d	f0 86	ea	a0	fa 77	83	a7	36																				
	33	24	48	4d af	70	2a	aa a3	da	b6	79 d8	a6	9e	2d																				
	54	38	20	ed a7	ff	fb	52 d3	ff	49	3f c3	ff	55	1e																				
	fb	ff	d9	7f 55	fe	ee	f2 08	5a	f3	12 08	86	88	a9																				
$\text{Compr}(IV_1, M_1)$														f0	20	48	6f	07	1b	f1	10	53	54	7a	86	f4	a7	15	3b	3c	95	0f	4b
Message 2																																	
IV_2	50	6b	01	78	ff	6d	18	91 a0	22	91	fd	3a	de	38	71	b2	c6	65	ea														
M_2	3f	44	38	38 81	ea	3d	ec a0	ea	a0	ee 51	83	a7	2c																				
	33	24	48	5d ab	70	2a	b6 6f	da	b6	6d d4	a6	9e	2f																				
	94	38	20	fd 13	ff	fb	4e ef	ff	49	3b 7f	ff	55	04																				
	db	ff	d9	6f 71	fe	ee	ee e4	5a	f3	06 04	86	88	ab																				
$\text{Compr}(IV_2, M_2)$														f0	20	48	6f	07	1b	f1	10	53	54	7a	86	f4	a7	15	3b	3c	95	0f	4b

the-art collision attack, we can project that a complete SHA-1 collision would take between 49 and 78 days on a 512 GPU cluster, and renting the equivalent GPU time on EC2 would cost between 75K US\$ and 120K US\$ and would plausibly take at most a few months.

Although freestart collisions do not directly translate to collisions for the hash function, they directly invalidate the security reduction of the hash function to the one of the compression function. Hence, obtaining a concrete example of such a collision further highlights the weaknesses of SHA-1 and existing users should quickly stop using this hash function. In particular, we believe that our work shows that the industry’s plan to move away from SHA-1 in 2017 might not be soon enough.

Outline. In Section 2, we provide our analysis and recommendations regarding the timeline of migration from SHA-1 to a secure hash function. In Section 3 we give a short description of the SHA-1 hash function and our notations. In Section 4, we explain the structure of our cryptanalysis and the various techniques used from a high level point of view, and we

GPU cards are comparable to NVidia Tesla cards and actually contain 2 physical GPU chips each. But due to their lower clock speed and slightly lower performance we estimate that each card is comparable to about one GTX970s.

later provide in Section 5 all the details of our attack for the interested readers.

2 Recommendations for the swift removal of SHA-1

Our work allowed to generate a freestart collision for the full SHA-1, but a collision for the entire hash algorithm is still unknown. There is no known generic and efficient algorithm that can turn a freestart collision into a plain collision for the hash function. However, the advances we have made do allow us to precisely estimate and update the computational and financial cost to generate such a collision with latest cryptanalysis advances [40] (the computational cost required to generate such a collision was actually a recurrent debate in the academic community since the first theoretical attack from Wang *et al.* [43]).

Schneier’s projections [35] on the cost of SHA-1 collisions in 2012 (on EC2: $\approx 700\text{K}$ US\$ by 2015, $\approx 173\text{K}$ US\$ by 2018 and $\approx 43\text{K}$ US\$ by 2021) were based on (an early announcement of) [40]. As mentioned earlier, these projections have been used to establish the timeline of migrating away from SHA-1-based signatures for secure Internet websites, resulting in a migration by January 2017 —one year before Schneier estimated that a SHA-1 collision would be within the resources of criminal syndicates.

However, as remarked in [17] and now further improved in this article thanks to the use of boomerang speed-up techniques [15], GPUs are much faster for this type of attacks (compared to CPUs) and we now precisely estimate that a full SHA-1 collision should not cost more than between 75K and 120K US\$ by renting Amazon EC2 cloud over a few months at the time of writing, in early autumn 2015. Our new GPU-based projections are now more accurate and they are significantly below Schneier’s estimations. More worrying, they are theoretically already within Schneier’s estimated resources of criminal syndicates as of today, almost two years earlier than previously expected, and one year before SHA-1 being marked as unsafe in modern Internet browsers. Therefore, we believe that migration from SHA-1 to the secure SHA-2 or SHA-3 hash algorithms should be done sooner than previously planned.

Note that it has previously been shown that a more advanced so-called chosen-prefix collision attack on MD5 allowed the creation of a rogue Certification Authority undermining the security of all secure websites [42]. Collisions on SHA-1 can result in *e.g.* signature forgeries, but do not directly undermine the security of the Internet at large. More advanced so-called chosen-prefix collisions [42] are significantly more threatening,

but currently much costlier to mount. Yet, given the lessons learned with the MD5 full collision break, it is not advisable to wait until these become practically possible.

At the time of the submission of this article in October 2015, we learned that in an ironic turn of events the CA/Browser Forum³ was planning to hold a ballot to decide whether to extend issuance of SHA-1 certificates through the year 2016 [10]. With our new cost projections in mind, we strongly recommended against this extension and the ballot was subsequently withdrawn [11]. Further action is also being considered by major browser providers such as Microsoft [23] and Mozilla [24] in speeding up the removal of SHA-1 certificates.

3 Preliminaries

3.1 Description of SHA-1

We start this section with a brief description of the SHA-1 hash function. We refer to the NIST specification document [26] for a more thorough presentation. SHA-1 is a hash function from the MD-SHA family which produces digests of 160 bits. It is based on the popular Merkle-Damgård paradigm [4,22], where the (padded) message input to the function is divided into k blocks of a fixed size (512 bits in the case of SHA-1). Each block is fed to a compression function h which then updates a 160-bit chaining value cv_i using the message block m_{i+1} , *i.e.* $cv_{i+1} = h(cv_i, m_{i+1})$. The initial value $cv_0 = IV$ is a predefined constant and cv_k is the output of the hash function.

Similarly to other members of the MD-SHA family, the compression function h is built around an *ad hoc* block cipher E used in a Davies-Meyer construction: $cv_{i+1} = E(m_{i+1}, cv_i) + cv_i$, where $E(x, y)$ is the encryption of the plaintext y with the key x and “+” denotes word-wise addition in $\mathbf{Z}/2^{32}\mathbf{Z}$. The block cipher itself is an 80-step (4 rounds of 20 steps each) five-branch generalized Feistel network using an Add-Rotate-Xor “ARX” step function. The internal state consists in five 32-bit registers $(A_i, B_i, C_i, D_i, E_i)$; at each step, a 32-bit extended message word W_i is

³ The CA/Browser Forum is the main association of industries regulating the use of digital certificates on the Internet.

used to update the five registers:

$$\begin{cases} A_{i+1} = (A_i \lll 5) + f_i(B_i, C_i, D_i) + E_i + K_i + W_i \\ B_{i+1} = A_i \\ C_{i+1} = B_i \ggg 2 \\ D_{i+1} = C_i \\ E_{i+1} = D_i \end{cases}$$

where K_i are predetermined constants and f_i are Boolean functions (see [Table 3-1](#) for their specifications). As all updated registers but A_{i+1} are just rotated copies of another, it is possible to equivalently express the step function in a recursive way using only the register A :

$$A_{i+1} = (A_i \lll 5) + f_i(A_{i-1}, A_{i-2} \ggg 2, A_{i-3} \ggg 2) + (A_{i-4} \ggg 2) + K_i + W_i.$$

Table 3-1. Boolean functions and constants of SHA-1

round	step i	$f_i(B, C, D)$	K_i
1	$0 \leq i < 20$	$f_{IF} = (B \wedge C) \oplus (\bar{B} \wedge D)$	0x5a827999
2	$20 \leq i < 40$	$f_{XOR} = B \oplus C \oplus D$	0x6ed6eba1
3	$40 \leq i < 60$	$f_{MAJ} = (B \wedge C) \oplus (B \wedge D) \oplus (C \wedge D)$	0x8fabbcdc
4	$60 \leq i < 80$	$f_{XOR} = B \oplus C \oplus D$	0xca62c1d6

Finally, the extended message words W_i are computed from the 512-bit message block, which is split into sixteen 32-bit words M_0, \dots, M_{15} . These sixteen words are then expanded linearly into the eighty 32-bit words W_i as follows:

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

The step function and the message expansion can both easily be inverted.

3.2 Differential collision attacks on SHA-1

We now introduce the main notions used in a collision attack on SHA-1 (and more generally on members of the MD-SHA family).

Background. In a differential collision attack on a (Merkle-Damgård) hash function, the goal of the attacker is to find a high-probability differential path (the differences being on the message, and also on the IV in the case of a freestart attack) which entails a zero difference on the final state of the function (*i.e.* the hash value). A pair of messages (and optionally IVs) following such a path indeed leads to a collision.

In the case of SHA-1 (and more generally ARX primitives), the way of expressing differences between messages is less obvious than for *e.g.* bit or byte-oriented primitives. It is indeed natural to consider both “XOR differences” (over \mathbf{F}_2^n) and “modular differences” (over $\mathbf{Z}/2^n\mathbf{Z}$) as both operations are used in the function. In practice, the literature on SHA-1 uses several hybrid representations of differences based on *signed XOR differences*. In its most basic form, such a difference is similar to an XOR difference with the additional information of the value of the differing bits for each message (and also of some bits equal in the two messages), which is a “sign” for the difference. This is an important information when one works with modular addition as the sign impacts the (absence of) propagation of carries in the addition of two differences. Let us for instance consider the two pairs of words $a = 11011000001_b$, $\hat{a} = 11011000000_b$ and $b = 10100111000_b$, $\hat{b} = 10100111001_b$; the XOR differences $(a \oplus \hat{a})$ and $(b \oplus \hat{b})$ are both 00000000001_b (which may be written $\dots\dots\dots\mathbf{x}$), meaning that $(a \oplus b) = (\hat{a} \oplus \hat{b})$. On the other hand, the signed XOR difference between a and \hat{a} may be written $\dots\dots\dots-$ to convey the fact that they are different on their lowest bit *and* that the value of this bit is 1 for a (and thence 0 for \hat{a}); similarly, the signed difference between b and \hat{b} may be written $\dots\dots\dots+$, which is a difference in the same position but of a different sign. From these differences, we can deduce that $(a + b) = (\hat{a} + \hat{b})$ because differences of different signs cancel; if we were to swap the values b and \hat{b} , both differences on a and b would have the same sign and indeed we would have $(a + b) \neq (\hat{a} + \hat{b})$ (though $(a \oplus b)$ and $(\hat{a} \oplus \hat{b})$ would still be equal). It is possible to extend signed differences to account for more generic combinations of possible values for each message bit; this was for instance done by De Cannière and Rechberger to aid in the automatic search of differential paths [6]. Another possible extension is to consider relations between various bits of the (possibly rotated) state words; this allows to efficiently keep track of the propagation of differences through the step function. Such differences are for instance used by Stevens [40], and also in this work (see Figure 5-2).

The structure of differential attacks on SHA-1 evolved to become quite specific. At a high level, they consist of: 1. a *non-linear* differential path

of low probability; 2. a *linear* differential path of high probability; 3. accelerating techniques.

The terms *non-linear* and *linear* refer to how the paths were obtained: the latter is derived from a linear (over \mathbf{F}_2^{32}) modelling of the step function. This kind of path is used in the probabilistic phase of the attack, where one simply tries many message pairs in order to find one that indeed “behaves” linearly. Computing the exact probability of this event is however not easy, although it is not too hard to find reasonable estimates. This probability is the main factor determining the final complexity of the attack.

The role of a non-linear path is to bootstrap the attack by bridging a state with no differences (the IV) with the start of the linear differential path⁴. In a nutshell, this is necessary because these paths do not typically lie in the kernel of the linearized **SHA-1**; hence it is impossible to obtain a collision between two messages following a fully linear path. This remains true in the present case of a freestart attack, even if the non-linear path now connects the start of the linear path with an IV containing some differences. Unlike the linear path, the non-linear one has a very low probability of being followed by random messages. However, the attacker can fully choose the messages to guarantee that they do follow the path, as he is free to set the 512 bits of the message. Hence finding conforming message pairs for this path effectively costs nothing in the attack.

Finally, the role of accelerating techniques is to find efficient ways of using the freedom degrees remaining after a pair following the non-linear path has been found, in order to delay the effective moment where the probabilistic phase of the attack starts.

We conclude this section with a short discussion of how to construct these three main parts of a (freestart) collision attack on **SHA-1**.

Linear path; local collisions. The linear differential paths used in collision attacks are built around the concept of *local collision*, introduced by Chabaud and Joux in 1998 to attack **SHA-0**. The idea underlying a local collision is first to introduce a difference in one of the intermediate

⁴ For the sake of simplicity, we ignore here the fact that a collision attack on **SHA-1** usually uses two blocks, with the second one having differences in its chaining value. The general picture is actually the following: once a pair of messages following the linear path \mathcal{P}^+ is found, the first block ends with a signed difference $+\Delta$; the sign of the linear path is then switched for the second block to become \mathcal{P}^- and following this path results in a difference $-\Delta$; the feedforward then cancels both differences and yields a collision.

state words of the function, say A_i , through a difference in the message word W_{i-1} . For an internal state made of j words ($j = 5$ in the case of SHA-0 or SHA-1), the attacker then uses subsequent differences in (possibly only some of) the message words $W_{i\dots i+(j-1)}$ in order to cancel any contribution of the difference in A_i in the computation of a new internal state $A_{i+1\dots i+j}$, which will therefore have no differences. The positions of these “correcting” differences are dictated by the step function, and there may be different options depending on the used Boolean function, though originally (and in most subsequent cases) these were chosen according to a linearized model (over \mathbf{F}_2^{32}) of the step functions.

Local collisions are a fit basis to generate differential paths of good probability. The main obstacle to do this is that the attacker does not control all of the message words, as some are generated by the message expansion. Chabaud and Joux showed how this could be solved by chaining local collisions along a *disturbance vector* (DV) in such a way that the final state of the function contains no difference and that the pattern of the local collisions is compatible with the message expansion. The disturbance vector just consists of a sparse message (of sixteen 32-bit words) that has been expanded with the linear message expansion of SHA-1. Every “one” bit of this expanded message then marks the start of a local collision (and expanding all the local collisions thus produces a complete linear path).

Each local collision in the probabilistic phase of the attack (roughly corresponding to the last three rounds) increases the overall complexity of the attack, hence one should use disturbance vectors that are sparse over these rounds. Initially, the evaluation of the probability of disturbance vector candidates was done mostly heuristically, using *e.g.* the Hamming weight of the vector ($[1,31,32,20,16]$), the sum of bit conditions for each local collision independently (not allowing carries) ($[45,46]$), and the product of independent local collision probabilities (allowing carries) ($[21,19]$). Manuel $[18,19]$ noticed that all disturbance vectors used in the literature belong to two classes $I(K, b)$ and $II(K, b)$. Within each class all disturbance vectors are forward or backward shifts in the step index (controlled by K) and/or bitwise cyclic rotations (controlled by b) of the same expanded message. We will use this notation through the remainder of this article.

Manuel also showed that success probabilities of local collisions are not always independent, causing biases in the above mentioned heuristic cost functions. This was later resolved by Stevens using a technique called joint local-collision analysis (JLCA) $[40,38]$, which allows to analyze entire

sets of differential paths over the last three rounds that conform to the (linear path entailed by the) disturbance vector. This is essentially an exhaustive analysis taking into account all local collisions together, using which one can determine the highest possible success probability. This analysis also produces a minimal set of sufficient conditions which, when all fulfilled, ensure that a pair of messages follows the linear path; the conditions are minimal in the sense that meeting all of them happens with this highest probability that was computed by the analysis. Although a direct approach is clearly unfeasible (as it would require dealing with an exponentially growing amount of possible differential paths), JLCA can be done practically by exploiting the large amount of redundancy between all the differential paths to a very large extent.

Non-linear differential path. The construction of non-linear differential paths was initially done by hand by Wang, Yin and Yu in their first attack on the full SHA-1 [43]. Efficient algorithmic construction of such differential paths was later proposed in 2006 by De Cannière and Rechberger, who introduced a guess-and-determine approach [6]. A different approach based on a meet-in-the-middle method was also proposed by Stevens *et al.* [38,14].

Accelerating techniques. For a given differential path, one can derive explicit conditions on state and message bits which are sufficient to ensure that a pair of messages follows the path. This lets the collision search to be entirely defined over a single compression function computation. Furthermore, they also allow detection of “bad” message pairs a few steps earlier compared to computing the state and verifying differences, allowing to abort computations earlier in this case.

An important contribution of Wang, Yin and Yu was the introduction of powerful *message modification* techniques, which followed an earlier work of Biham and Chen who introduced *neutral bits* to produce better attacks on SHA-0 [1]. The goal of both techniques is for the attacker to make a better use of the available freedom in the message words in order to decrease the complexity of the attack. Message modifications try to correct bad message pairs that only slightly deviate from the differential path, and neutral bits try to generate several good message pairs out of a single one (by changing the value of a bit which does not invalidate nearby sufficient conditions with good probability). In essence, both techniques allow to amortize part of the computations, which effectively delays the beginning of the purely probabilistic phase of the attack.

Finally, Joux and Peyrin showed how to construct powerful neutral bits and message modifications by using auxiliary differential paths akin to *boomerangs* [15], which allow more efficient attacks. In a nutshell, a boomerang (in collision attacks) is a small set of bits that together form a local collision. Hence flipping these bits together ensures that the difference introduced by the first bit of the local collision does not propagate to the rest of the state; if the initial difference does not invalidate a sufficient condition, this local collision is indeed a neutral bit. Yet, because the boomerang uses a single (or sometimes a few) local collision, more differences will actually be introduced when it goes through the message expansion. The essence of boomerangs is thus to properly choose where to locate the local collisions so that no differences are introduced for the most steps possible.

4 Attack overview

In this section we provide an overview of how our attack was constructed. At a high level, it consists of the following steps:

1. *Disturbance vector selection*: We need to select the best disturbance vector for our attack. This choice is based on results provided by joint local collision analysis (JLCA), taking into account constraints on the number and position of sufficient conditions on the IV implied by the disturbance vector. We explain this in [Section 4.1](#).
2. *Finding optimal attack conditions*: Having selected a disturbance vector, we need to determine a set of attack conditions over all steps consisting of sufficient conditions for state bits up to some step, augmented by message bit relations. We use non-linear differential path construction methods to determine conditions within the first round. Using JLCA we derive an optimal complete set of attack conditions that given the first round path leads to the highest possible success probability over all steps, yet minimizes the number of conditions within this model. We detail this in [Section 4.2](#).
3. *Finding and analyzing boomerangs and neutral bits*: To speed up the freestart collision attack, we exploit advanced message modification techniques such as (multiple) neutral bits and boomerangs. In order to find suitable candidates, we sample partial solutions fulfilling the above attack conditions up to an early step. The samples are used to test many potential boomerangs and neutral bits, and only ones of good quality and that do not introduce contradictions will be used. In

particular, no boomerang or neutral bit may invalidate the attack conditions of the so-called base solution (see below, includes all message bit relations) with non-negligible probability. We also use sampling to estimate the probability of interaction between boomerang and neutral bits with particular sufficient conditions, in the forward and backward direction. Although we do not allow significant interaction in the backward direction, we use these probabilities to determine at which step the boomerang or neutral bit are used. This is explained in [Section 4.3](#).

4. *Base solution generation*: Before we can apply neutral bits and boomerangs, we first need to compute a partial solution over 16 consecutive steps. Only this partial solution can then be extended to cover more steps by using neutral bits and boomerangs. We call such a solution a *base solution*; it consists of state words A_{-3}, \dots, A_{17} and message words W_1, \dots, W_{16} . The cost for generating base solutions is relatively low compared to the overall attack cost, therefore it is not heavily optimized and the search is run on regular CPUs. This is further explained in [Section 4.4](#).
5. *Application of neutral bits and boomerangs on GPU*: We extend each base solution into solutions over a larger number of steps by successively applying neutral bits and boomerangs and verifying sufficient conditions. Once all neutral bits and boomerangs have been exploited, the remainder of the steps have to be fulfilled probabilistically. This is computationally the most intensive part, and it is therefore implemented on GPUs that are significantly more cost-efficient than CPUs, using the highly efficient framework introduced by Karpman, Peyrin and Stevens [17]. More details are provided in [Section 4.5](#).

All these steps strongly build upon the continuous series of papers that have advanced the state-of-the-art in **SHA-1** cryptanalysis, yet there are still small adaptations and improvements used for this work. We now describe all these points in more details.

4.1 Disturbance vector selection

It is possible to compute exactly the highest success probability over the linear part by using joint-local collision analysis [40]. By further using the improvements described in [17], one can restrict carries for the steps where sufficient conditions are used and obtain the sufficient conditions for those steps immediately.

The number of sufficient conditions at the beginning of round 2 and the associated highest success probability for the remaining steps provide

Table 4-1. Disturbance vector analysis. For each DV, under $c_{[24,80]}$, we list the negative \log_2 of the success probability over steps [24, 80) assuming that all sufficient conditions up to A_{24} have been satisfied. The columns c_{23} and c_{22} list the number of conditions on A_{24} (in step 23) and A_{23} (in step 22), respectively. The final column represents an estimated runtime in days on a cluster consisting of 64 GTX970s based on $c_{[24,80]}$.

DV	Cost $c_{[24,80]}$	Cost c_{23}	Cost c_{22}	Days on 64 GPUs
I(48,0)	61.6	1	3	39.1
I(49,0)	60.5	3	2	18.3
I(50,0)	61.7	2	1	41.8
I(51,0)	62.1	1	2	55.7
I(48,2)	64.4	1	2	281.9
I(49,2)	62.8	2	3	90.4
II(46,0)	64.8	1	0	369.5
II(50,0)	59.6	1	2	9.9
II(51,0)	57.5	3	3	2.2
II(52,0)	58.3	3	3	4.1
II(53,0)	59.9	3	2	11.8
II(54,0)	61.3	2	1	31.4
II(55,0)	60.7	1	3	21.0
II(56,0)	58.9	3	2	6.3
II(57,0)	59.3	2	3	7.9
II(58,0)	59.7	3	2	10.5
II(59,0)	61.0	3	2	26.2
II(49,2)	61.0	2	3	26.1
II(50,2)	59.4	3	2	8.7
II(51,2)	59.4	2	3	8.5

insight into the attack complexity under different circumstances. In [Table 4-1](#) we give our analysis results for various DVs, listing the negative \log_2 of the success probability over steps $[24, 80)$ assuming that all sufficient conditions up to A_{24} have been satisfied; we also include the number of conditions on A_{24} and A_{23} . The final column represents an estimated runtime in days on a cluster consisting of 64 GTX970s based on $c_{[24,80)}$, by multiplying the runtime of the 76-step freestart GPU attack [17] with the difference between the costs $c_{[24,80)}$ for the 76-step attack and the DVs in the table.

Considering [Table 4-1](#), the obvious choice of DV to mount a full collision attack on SHA-1 would be $\text{II}(51,0)$. However in the present case of a freestart attack additional constraints need to be taken into account. In particular the choice of the DV determines the possible differences in the IV, as these have to cancel the differences of the final state $(A_{80}, B_{80}, C_{80}, D_{80}, E_{80})$. This impacts the estimated runtime as follows: if there are sufficient conditions present on A_0 then the estimated runtime in the last column should be multiplied by $2^{c_{23}}$. Indeed, the 76-step freestart attack did not have any sufficient conditions on A_0 and could thus ignore step 0, leading to an offset of one in the probabilistic phase. Moreover, if the IV differences are denser or ill-located (compared to the 76-step attack), then more neutral bits and boomerangs are likely to interact badly with the sufficient conditions on the IV, when propagated backwards. If only few neutral bits and boomerangs can be used for a given DV, the cost of the attack would rise significantly. The number of conditions c_{23} and c_{22} for each DV allow to estimate how much more expensive a vector will be in the case where the probabilistic phase effectively starts sooner than in step A_{25} as for the 76-step attack.

Taking the freestart setting into account, and with a preliminary analysis of available neutral bits and boomerangs, the best option eventually seemed to be $\text{II}(59,0)$. This DV is actually a downward shift by four of $\text{II}(55,0)$, which was used in the 76-step attack. Consequently, this choice leads to the same IV sufficient conditions as in the latter.

4.2 Finding optimal attack conditions

Using joint local collision analysis, we could obtain sufficient conditions for the beginning of the second round and IV differences that are optimal (*i.e.*, with the highest probability of cancelling differences in the final state). What remains to do is to construct a non-linear differential path for the first round. For this, we used the meet-in-the-middle method using the public HashClash implementation [14]. Although we tried both non-linear

differential path construction methods, *i.e.* guess-and-determine, using our own implementation, and the meet-in-the-middle method, we have found that the meet-in-the-middle approach generally resulted in fewer conditions. Furthermore, the control on the position of these conditions was greater with the meet-in-the-middle approach.

This differential path for the first round was then used as input for another run of joint local collision analysis. In this case the run was over all 80 steps, also replacing the differences assumed from the disturbance vector with the differences in the state words coming from the non-linear path of the first round; switching the sign of a difference was also allowed when it resulted in a sparser overall difference. In this manner joint local collision analysis is able to provide a complete set of attack conditions (*i.e.*, sufficient conditions for the state words and linear relations on message bits) that is optimized for the highest success probability over the last three rounds, all the while minimizing the amount of conditions needed.

In fact, JLCA outputs many complete sets that only vary slightly in the signing of the differences. For our selected disturbance vector $\text{II}(59,0)$ it turned out that this direct approach is far too costly and far too memory-consuming, as the amount of complete sets grows exponentially with the number of steps for which we desired sufficient conditions sets. We were able to improve this by introducing attack condition classes, where two sets of sufficient conditions belong to the same class if their sufficient conditions over the last five state words are identical. By expressing the attack condition classes over steps $[0, i]$ as extensions of attack conditions classes over steps $[0, i - 1]$, we only have to work with a very small number of class representatives at each step, making it very practical.

Note that we do not exploit this to obtain additional freedom for the attack yet, However, it allows us to automatically circumvent random unpredictable contradictions between the attack conditions in the densest part, by randomly sampling complete sets until a usable one is found. We previously used the guess-and-determine approach to resolve such contradictions by changing signs, however this still required some manual interaction.

The resulting sufficient conditions on the state are given in the form of the differential path in [Figure 5-1](#) (using the symbols of [Figure 5-2](#)) and the message bit relations are given in [Figure 5-3](#) through [Figure 5-5](#).

4.3 Finding and analyzing neutral bits and boomerangs

Generating and analyzing the boomerangs and neutral bits used in the attack was done entirely automatically as described below. This process depends on a parameter called the *main block offset* (specific to a freestart attack) that determines the offset of the message freedom window used during the attack. We have selected a main block offset of 5 as this led to the best distribution of usable neutral bits and boomerangs. This means that all the neutral bits and boomerangs directly lead to changes in the state from steps 5 up to 20, and that these changes propagate to steps 4 down to 0 backwards and steps 21 up to 79 forwards.

Because the dense area of the attack conditions may implicitly force certain other bits to specific values (resulting in hidden conditions), we use more than 4000 sampled solutions for the given attack conditions (over steps 1 up to 16) in the analysis. The 16 steps fully determine the message block, and also verify the sufficient conditions in the IV and in the dense non-linear differential path of the first round. It should be noted that for this step it is important to generate every sample independently. Indeed using *e.g.* message modification techniques to generate many samples from a single one would result in a biased distribution where many samples would only differ in the last few steps.

Boomerang analysis. We analyze potential boomerangs that flip a single state bit together with 3 or more message bits. Each boomerang should be orthogonal to the attack conditions, *i.e.*, the state bit should be free of sufficient conditions, while flipping the message bits should not break any of the message bit relations (either directly or through the message propagation). Let $t \in [6, 16]$, $b \in [0, 31]$ be such that the state bit $A_t[b]$ has no sufficient condition.

First, we determine the best usable boomerang on $A_t[b]$ as follows. For every sampled solution, we flip that state bit and compute the signed bit differences between the resulting and the unaltered message words W_5, \dots, W_{20} . We verify that the boomerang is usable by checking that flipping its constituting bits breaks none of the message bit relations. We normalize these signed bit differences by negating them all when the state bit is flipped from 1 to 0. In this manner we obtain a set of usable boomerangs for $A_t[b]$. We determine the auxiliary conditions on message bits and state bits and only keep the best usable boomerang that has the fewest auxiliary conditions.

Secondly, we analyze the behaviour of the boomerang over the backwards steps. For every sampled solution, we simulate the application of

the boomerang by flipping the bits of the boomerang. We then recompute steps 4 to 0 backwards and verify if any sufficient condition on these steps is broken. Any boomerang that breaks any sufficient conditions on the early steps with probability higher than 0.1 is dismissed.

Thirdly, we analyze the behaviour of the boomerang over the forward steps. For every sampled solution, we simulate the application of the boomerang by flipping its constituting bits. We then recompute steps 21 up to 79 forwards and keep track of any sufficient condition for the differential path that becomes violated. A boomerang will be used at step i in our attack if it does not break any sufficient condition up to step $i - 1$ with probability more than 0.1.

Neutral bits analysis. The neutral bit analysis uses the same overall approach as the boomerangs, with the following changes. After boomerangs are determined, their conditions are added to the previous attack conditions and used to generate a new set of solution samples. Usable neutral bits consist of a set of one or more message bits that are flipped simultaneously. However, unlike for boomerangs, the reason for flipping more than one bit is to preserve message bit relations, and not to control the propagation of a state difference. Let $t \in [5, 20]$, $b \in [0, 31]$ be a candidate neutral bit; flipping $W_t[b]$ may possibly break certain message bit relations. We express each message bit relation over W_5, \dots, W_{20} using linear algebra, and use Gaussian elimination to ensure that each of them has a unique last message bit $W_i[j]$ (*i.e.* where $i * 32 + j$ is maximal). For each relation involving $W_t[b]$, let $W_i[j]$ be its last message bit. If (i, j) equals (t, b) then this neutral bit is not usable (indeed, it would mean that its value is fully determined by earlier message bits). Otherwise we add bit $W_i[j]$ to be flipped together with $W_t[b]$ as part of the neutral bit. Similarly to boomerangs, we dismiss any neutral bit that breaks sufficient conditions backwards with probability higher than 0.1. The step i in which the neutral bit is used is determined in the same way as for the boomerangs.

The boomerangs we have selected are given in [Figure 5-7](#) and the neutral bits are listed in [Figure 5-6](#). In the case of the latter, only the first neutral bit is given and not the potential corrections for the message bit relations.

4.4 Base solution generation on CPU

We are now equipped with a set of attack conditions, including some that were added by the selected boomerangs and neutral bits. However, before

these can be applied, we first need to compute partial solutions over 16 consecutive steps. Since the selected neutral bits and boomerangs cannot be used to correct the sufficient conditions on the IV, these have to be pre-satisfied as well. Therefore, we compute what we call *base solutions* over steps $1, \dots, 16$ that fulfill all state conditions on A_{-4}, \dots, A_{17} and all message bit relations. A base solution itself consists of state words A_{-3}, \dots, A_{17} and message words W_1, \dots, W_{16} (although the implementation of the GPU step implies that the message words are translated to the equivalent message words W_5, \dots, W_{20} with the main block offset of 5).

The C++ code generating base solutions is directly compiled from the attack and auxiliary conditions. In this manner, all intermediate steps and condition tables can be hard-coded, and we can apply some static local optimizations eliminating unnecessary computations where possible. However, we do not exploit more advanced message modification techniques within these first 16 steps yet.

Generating the base solutions only represents a small part of the cost of the overall attack, and it is run entirely on CPU. Although theoretically we need only a few thousand base solutions to be successful given the total success probability over the remaining steps and the remaining freedom degrees yet to be used, in practice we need to generate a small factor more to ensure that all GPUs have enough work.

4.5 Applying neutral bits and boomerangs on GPU

We now describe the final phase of the attack, which is also the most computationally intensive; as such, it was entirely implemented on GPUs. In particular, we used 65 recent Nvidia GTX970 [30] GPUs that feature 1664 small cores operating at a clock speed of about 1.2GHz; each card cost about 350 US\$ in 2015.⁵ In [17], the authors evaluate a single GTX970 to be worth 322 CPU cores⁶ for raw SHA-1 operations, and about 140 cores for their SHA-1 attack.

We make use of the same efficient framework for Nvidia GPUs [17]. This makes use of the CUDA toolkit that provides programming extensions to C and C++ for convenient programming. For each step of SHA-1 wherein we use neutral bits and boomerangs, there will be a separate GPU-specific C++ function. Each function will load solutions up to that step from a global cyclic buffer; extend those solutions using the freedom

⁵ With the right motherboard one can place up to 4 such GPUs on a single machine.

⁶ Intel Haswell Core-i5 3.2GHz CPU.

for that step by triggering the available neutral bits or boomerangs; verify the sufficient conditions; and finally save the resulting partial solution extended by one step in the next global cyclic buffer. The smallest unit that can act independently on Nvidia GPUs is the *warp*, which consists of 32 threads that can operate on different data, but should execute the same instruction for best performance. When threads within a warp diverge onto different execution paths, these paths are executed serially, not in parallel. In the framework, the threads within each warp will agree on which function (thus which step) to execute together, resulting in reads, computations, and conditional writes that are coherent between all threads of the warp. We refer the reader to the original paper introducing this framework for a more detailed description [17].

The exact details of which neutral bits and which boomerangs are used for each step are given in [Section 5](#).

In the probabilistic phase, after all freedom degrees have been exhausted, we can verify internal state collisions that should happen after steps 39 and 59 (for a message pair that follows the differential path), as these are steps with no active differences in the disturbance vector. These checks are still done on the GPU. Solutions up to A_{60} are passed back to the CPU for further verification to determine if a complete freestart collision has been found.

We would like to note that in comparison with the attack on 76 steps, this work introduces boomerangs and has a slightly bigger count of neutral bits (60 v. 51). As a result, this required to use more intermediate buffers, and consequently a slightly more careful management of the memory. Additionally, in this work there is a relatively high proportion of the neutral bits that need additional message bits to be flipped to ensure no message bit relation is broken, whereas this only happens twice in the 76-step attack. These two factors result in an attack that is slightly more complex to implement, although neither point is a serious issue.

5 Attack details

5.1 Sufficient conditions

We give a graphical representation of the differential path used in our attack up to step 28 in [Figure 5-1](#), consisting of sufficient conditions for the state, and the associated message signed bit differences. The meaning of the bit condition symbols are defined in [Figure 5-2](#). Note that the signs of message bit differences are enforced through message bit relations. All message bit relations used in our attack are given in [Figure 5-3](#)

through [Figure 5-5](#). The remainder of the path can easily be determined by linearization of the step function given the differences in the message.

A-4:		
A-3:		
A-2:		
A-1:	1...1...0.....+	
A0 :	01..0...1.....	W0 : x.+...+
A1 :	11+^...++....	W1 : ..-...
A2 :	..-11-1. 1.....1+1 10.1.0..	W2 : ..+...-
A3 :	.0.0-001 1.^..10..	..+01.011 11^0.1.1	W3 : ..-...-
A4 :	.1.11+-1 +^^+1^^	^011^^- +++++-+	W4 :
A5 :	.+...-++ ++++++	+++++ +0-1111	W5 :
A6 :	.0.0.1.0 11.111.1	1110-010 0-1.10+	W6 : x+...+
A7 :	1-+.1.0 10100010	00000011 1+-.0.+	W7 : ...-+
A8 :	0+.0.0...0. .+-.0.1	W8 : x-.....
A9 :	.+..0.0...0.+...^	W9 : x.-+...
A10:	.+.....+0..	W10: ..+...
A11:	...-.....	W11: x.++++
A12:	...0.1...1..	W12: ..-.....
A13:	.1...0...!^	W13: ..+...+
A14:	+.....	W14: x++.+...
A15:	1.1-.....!	W15:+...
A16:	+..10.1...	W16: x+.....
A17:	1-..0...^	W17: x.+++.
A18:	.-.0...!	W18: ..+...-
A19:	+.s.....	W19: x.+---
A20:	-...R...	W20: x.++.....
A21:	-..R.....	W21:
A22:	-...S...^	W22: x.---
A23:	-...R...	W23:-
A24:	-..rs.....	W24: .-+---
A25:	-..r.....	W25:+
A26:	-...s...	W26: .+---
A27:	-..r.....	W27: x.++-
A28:	W28: x+-.-
A29:	..-.....	

Fig. 5-1. The differential path used in the attack up to step 28. The meaning of the different symbols is given in [Figure 5-2](#)

5.2 The neutral bits

We give here the list of the neutral bits used in our attack. There are 60 of them over the 7 message words W_{14} to W_{20} , distributed as follows:

- W_{14} : 6 neutral bits at bit positions (starting with the least significant bit (*LSB*) at zero) 5,7,8,9,10,11
- W_{15} : 11 neutral bits at positions 4,7,8,9,10,11,12,13,14,15,16
- W_{16} : 9 neutral bits at positions 8,9,10,11,12,13,14,15,16
- W_{17} : 10 neutral bits at positions 10,11,12,13,14,15,16,17,18,19

<i>Symbol</i>	<i>Condition on $(\mathbf{A}_t[\mathbf{i}], \mathbf{A}'_t[\mathbf{i}])$</i>
.	$A_t[i] = A'_t[i]$
x	$A_t[i] \neq A'_t[i]$
+	$A_t[i] = 0, \quad A'_t[i] = 1$
-	$A_t[i] = 1, \quad A'_t[i] = 0$
0	$A_t[i] = A'_t[i] = 0$
1	$A_t[i] = A'_t[i] = 1$
~	$A_t[i] = A'_t[i] = A_{t-1}[i]$
!	$A_t[i] = A'_t[i] \neq A_{t-1}[i]$
r	$A_t[i] = A'_t[i] = (A_{t-1} \ggg 2)[i]$
R	$A_t[i] = A'_t[i] \neq (A_{t-1} \ggg 2)[i]$
s	$A_t[i] = A'_t[i] = (A_{t-2} \ggg 2)[i]$
S	$A_t[i] = A'_t[i] \neq (A_{t-2} \ggg 2)[i]$

Fig. 5-2. Bit conditions

- W_{18} : 11 neutral bits at positions 4,6,7,8,9,10,11,12,13,14,15
- W_{19} : 8 neutral bits at positions 6,7,8,9,10,11,12,14
- W_{20} : 5 neutral bits at positions 6,11,12,13,15

We give a graphical representation of the position of these neutral bits in [Figure 5-6](#).

Not all of the neutral bits of the same word (say W_{14}) are used at the same step during the attack. Their repartition in that respect is as follows

- Bits neutral up to step 18 (excluded): $W_{14}[8,9,10,11]$, $W_{15}[13,14,15,16]$
- Bits neutral up to step 19 (excluded): $W_{14}[5,7]$, $W_{15}[8,9,10,11,12]$, $W_{16}[12,13,14,15,16]$
- Bits neutral up to step 20 (excluded): $W_{15}[4,7,8,9]$, $W_{16}[8,9,10,11,12]$, $W_{17}[12,13,14,15,16]$
- Bits neutral up to step 21 (excluded): $W_{17}[10,11,12,13]$, $W_{18}[15]$
- Bits neutral up to step 22 (excluded): $W_{18}[9,10,11,12,13,14]$, $W_{19}[10,14]$
- Bits neutral up to step 23 (excluded): $W_{18}[4,6,7,8]$, $W_{19}[9,11,12]$, $W_{20}[15]$
- Bits neutral up to step 24 (excluded): $W_{19}[6,7,8]$, $W_{20}[11,12,13]$
- Bit neutral up to step 25 (excluded): $W_{20}[7]$

One should note that this list only includes a single bit per neutral bit group. As we mentioned in the previous section, some additional flips may be needed in order to preserve message bit relations.

- W0[4] = 0	- W9[29] = 1	- W18[27] = 1	- W29[28] = 0
- W0[25] = 0	- W10[2] = 1	- W18[29] = 0	- W29[29] = 0
- W0[29] = 0	- W10[26] = 0	- W19[3] = 0	- W30[27] ^ W30[28] = 1
- W1[2] = 0	- W10[27] = 0	- W19[4] = 1	- W30[30] = 1
- W1[3] = 0	- W10[28] = 0	- W19[26] = 1	- W31[2] = 0
- W1[4] = 1	- W10[29] = 1	- W19[27] = 1	- W31[3] = 0
- W1[26] = 1	- W11[1] = 0	- W19[28] = 1	- W31[28] = 0
- W1[29] = 1	- W11[3] = 0	- W19[29] = 0	- W31[29] = 0
- W2[2] = 0	- W11[4] = 1	- W20[4] = 0	- W33[28] ^ W33[29] = 1
- W2[4] = 1	- W11[26] = 0	- W20[28] = 0	- W30[4] ^ W34[29] = 0
- W2[25] = 1	- W11[27] = 0	- W20[29] = 0	- W35[27] = 0
- W2[26] = 1	- W11[28] = 0	- W21[2] = 0	- W35[28] = 0
- W2[29] = 0	- W11[29] = 0	- W21[3] = 0	- W35[4] ^ W39[29] = 0
- W3[1] = 1	- W12[4] = 1	- W22[4] = 0	- W58[29] ^ W59[29] = 0
- W3[3] = 0	- W12[29] = 1	- W22[27] = 1	- W57[29] ^ W59[29] = 0
- W3[4] = 1	- W13[2] = 0	- W22[28] = 1	- W55[4] ^ W59[29] = 0
- W3[25] = 1	- W13[3] = 0	- W22[29] = 1	- W53[29] ^ W54[29] = 0
- W3[26] = 1	- W13[4] = 1	- W23[3] = 1	- W52[29] ^ W54[29] = 0
- W3[29] = 1	- W13[26] = 0	- W23[4] = 0	- W51[28] ^ W51[29] = 1
- W4[4] = 0	- W13[29] = 0	- W23[27] = 1	- W50[4] ^ W54[29] = 0
- W5[2] = 0	- W14[2] = 0	- W24[4] = 0	- W50[28] ^ W51[28] = 0
- W5[3] = 0	- W14[4] = 1	- W24[27] = 1	- W50[29] ^ W51[28] = 1
- W5[4] = 0	- W14[26] = 1	- W24[28] = 1	- W49[28] ^ W51[28] = 0
- W5[26] = 1	- W14[27] = 0	- W24[29] = 0	- W48[29] ^ W48[30] = 0
- W6[2] = 0	- W14[29] = 0	- W24[30] = 1	- W47[3] ^ W51[28] = 0
- W6[4] = 1	- W14[30] = 0	- W26[4] = 0	- W47[4] ^ W51[28] = 1
- W6[26] = 0	- W15[1] = 0	- W26[28] = 1	- W46[29] ^ W51[28] = 1
- W6[27] = 0	- W15[26] = 1	- W26[29] = 1	- W45[4] ^ W51[28] = 0
- W6[30] = 0	- W15[27] = 0	- W26[30] = 0	- W44[29] ^ W51[28] = 0
- W7[1] = 0	- W16[4] = 1	- W27[2] = 1	- W43[4] ^ W51[28] = 1
- W7[26] = 0	- W16[30] = 0	- W27[3] = 0	- W43[29] ^ W51[28] = 0
- W7[27] = 1	- W17[2] = 1	- W27[4] = 0	- W41[4] ^ W51[28] = 0
- W8[4] = 0	- W17[3] = 1	- W27[27] = 0	- W63[4] ^ W67[29] = 0
- W8[30] = 1	- W17[4] = 0	- W27[28] = 1	- W79[5] = 0
- W9[2] = 0	- W17[26] = 0	- W27[29] = 0	- W78[0] = 1
- W9[3] = 0	- W17[28] = 0	- W28[27] = 0	- W77[1] ^ W78[6] = 1
- W9[4] = 1	- W17[29] = 0	- W28[29] = 1	- W75[5] ^ W79[30] = 0
- W9[26] = 1	- W18[2] = 1	- W28[30] = 0	- W74[0] ^ W79[30] = 1
- W9[28] = 0	- W18[26] = 1	- W29[2] = 0	

Fig. 5-3. The message bit-relations used in the attack.

5.3 The boomerangs

We finally give the boomerangs used in the attack, which are regrouped in two sets of two. The first one first introduces a difference in the message on word W_{10} ; as it does not significantly impact conditions up to step 27, it is used to increase the number of partial solutions A_{28} that are generated. The second set first introduces a difference on word W_{11} , and is used to generate partial solutions at A_{30} . More precisely, the four boomerangs have their first differences at bits 7,8 of W_{10} and 8,9 of W_{11} . In [Figure 5-7](#), we give a graphical representation of the complete set of message bits to be flipped for each boomerang. One can see that these indeed follow the pattern of a local collisions.


```

W40: . . . . .
W41: . . . . . w . . . . .
W42: . . . . . u . . . . .
W43: . . v . . . . . u . . . . .
W44: . . t . . . . . s . . . . .
W45: . . . . . s . . . . .
W46: . . r . . . . . p . . . . .
W47: . . . . . q p . . . . .
W48: . . o o . . . . .
W49: . . . n . . . . .
W50: . . m l . . . . . k . . . . .
W51: . . J @ . . . . .
W52: . . i . . . . .
W53: . . h . . . . .
W54: . . * . . . . .
W55: . . . . . g . . . . .
W56: . . . . .
W57: . . f . . . . .
W58: . . e . . . . .
W59: . . $ . . . . .
W60: . . . . .
W61: . . . . .
W62: . . . . .
W63: . . . . . x . . . . .
W64: . . . . .
W65: . . . . .
W66: . . . . .
W67: . . x . . . . .
W68: . . . . .
W69: . . . . .
W70: . . . . .
W71: . . . . .
W72: . . . . .
W73: . . . . .
W74: . . . . . a
W75: . . . . . z . . . . .
W76: . . . . .
W77: . . . . . y . . . . .
W78: . . . . . Y . . . . . 1
W79: . . % . . . . . 0 . . . . .
@ = j l M n p Q R s t U v w
* = h i k
$ = e f g
% = z A

```

Fig. 5-5. The message bit-relations used in the attack for words W_{40} to W_{79} (graphical representation, continued). Non-alphanumeric symbols are used as shorthand for bit positions with more than one relation.

```

W14: ..... .xxxx x.x.....
W15: .....x xxxxxxxx x..x....
W16: .....x xxxxxxxx .....
W17: .....xxxx xxxxxx.. .....
W18: ..... xxxxxxxx xx.x....
W19: ..... .x.xxxxx xx.....
W20: ..... x.xxx... .x.....

```

Fig. 5-6. The 60 neutral bits. An “x” represents the presence of a neutral bit, and a “.” the absence thereof. The LSB position is the rightmost one.

```

W10: .....BA .....
W11: .....ba....D C.....
W12: .....dc.... .....
W13: .....
W14: .....a.....
W15: .....ba.....
W16: .....dc.....

```

Fig. 5-7. The local collision patterns for each of the four boomerangs. The position of the first difference to be introduced is highlighted with a capital letter; the correcting differences must then have a sign different from this one. Note that boomerang “A” uses one more difference than the others.

Software disclosure policy

To allow verification and improve understanding of our new results, we intend to release our engineered freestart attack code for graphic cards at <https://sites.google.com/site/itstheshappening/>.

However, this source code does not directly enable the engineering of a SHA-1 collision attack. Any cryptanalytic tools needed for engineering a full SHA-1 collision attack will be released independently in a responsible manner.

Acknowledgements

We would like to express our gratitude to Orr Dunkelman for the use of his cluster with NVidia Tesla K10 cards. We also thank the anonymous reviewers for their helpful comments.

References

1. 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)
2. Biham, E., Chen, R., Joux, A., Carribault, P., Lemuet, C., Jalby, W.: Collisions of SHA-0 and Reduced SHA-1. In: Cramer, R. (ed.) EUROCRYPT. Lecture Notes in Computer Science, vol. 3494, pp. 36–57. Springer (2005)
3. 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)
4. Damgård, I.: A Design Principle for Hash Functions. In: Brassard, G. (ed.) CRYPTO. Lecture Notes in Computer Science, vol. 435, pp. 416–427. Springer (1989)
5. 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)
6. 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)
7. 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)
8. den Boer, B., Bosselaers, A.: Collisions for the Compression Function of MD5. In: Helleseeth, T. (ed.) EUROCRYPT. Lecture Notes in Computer Science, vol. 765, pp. 293–304. Springer (1993)
9. Dobbertin, H.: Cryptanalysis of MD4. In: Gollmann, D. (ed.) FSE. Lecture Notes in Computer Science, vol. 1039, pp. 53–69. Springer (1996)
10. Forum, C.: Ballot 152 - Issuance of SHA-1 certificates through 2016. Cabforum mailing list (2015)

11. Forum, C.: Ballot 152 - Issuance of SHA-1 certificates through 2016. Cabforum mailing list (2015)
12. Grechnikov, E.A.: Collisions for 72-step and 73-step SHA-1: Improvements in the Method of Characteristics. IACR Cryptology ePrint Archive 2010, 413 (2010)
13. Grechnikov, E.A., Adinets, A.V.: Collision for 75-step SHA-1: Intensive Parallelization with GPU. IACR Cryptology ePrint Archive 2011, 641 (2011)
14. Hashclash project webpage, <https://marc-stevens.nl/p/hashclash/>
15. 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)
16. Jutla, C.S., Patthak, A.C.: A matching lower bound on the minimum weight of sha-1 expansion code. Cryptology ePrint Archive, Report 2005/266 (2005)
17. 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), <http://dx.doi.org/10.1007/978-3-662-47989-6>
18. Manuel, S.: Classification and generation of disturbance vectors for collision attacks against sha-1. Cryptology ePrint Archive, Report 2008/469 (2008)
19. Manuel, S.: Classification and generation of disturbance vectors for collision attacks against SHA-1. Des. Codes Cryptography 59(1-3), 247–263 (2011)
20. Matusiewicz, K., Pieprzyk, J.: Finding good differential patterns for attacks on SHA-1. In: Ytrehus, Ø. (ed.) Coding and Cryptography, International Workshop, WCC 2005. Lecture Notes in Computer Science, vol. 3969, pp. 164–177. Springer (2005)
21. 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)
22. Merkle, R.C.: One Way Hash Functions and DES. In: Brassard, G. (ed.) CRYPTO. Lecture Notes in Computer Science, vol. 435, pp. 428–446. Springer (1989)
23. Microsoft: SHA-1 Deprecation Update. Microsoft blog (2015)
24. Mozilla: Continuing to Phase Out SHA-1 Certificates. Mozilla Security Blog (2015)
25. National Institute of Standards and Technology: FIPS 180: Secure Hash Standard (May 1993)
26. National Institute of Standards and Technology: FIPS 180-1: Secure Hash Standard (April 1995)
27. National Institute of Standards and Technology: FIPS 180-2: Secure Hash Standard (August 2002)
28. National Institute of Standards and Technology: Special Publication 800-57 - Recommendation for Key Management Part 1: General (Revision 3) (July 2012)
29. National Institute of Standards and Technology: FIPS 202: SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions (August 2015)
30. Nvidia Corporation: Nvidia Geforce GTX 970 Specifications. <http://www.geforce.com/hardware/desktop-gpus/geforce-gtx-970/specifications>
31. 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)
32. Rijmen, V., Oswald, E.: Update on SHA-1. In: Menezes, A. (ed.) CT-RSA. Lecture Notes in Computer Science, vol. 3376, pp. 58–71. Springer (2005)

33. 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)
34. Rivest, R.L.: RFC 1321: The MD5 Message-Digest Algorithm (April 1992)
35. Schneier, B.: When will we see collisions for sha-1? Schneier on Security (2012)
36. Services, A.W.: Amazon EC2 – Virtual Server Hosting. aws.amazon.com (Retrieved Jan 2016)
37. Survey of the ssl implementation of the most popular web sites. TIM Trustworthy Internet Movement (2015)
38. Stevens, M.: Attacks on Hash Functions and Applications. Ph.D. thesis, Leiden University (June 2012)
39. Stevens, M.: Counter-Cryptanalysis. In: Canetti, R., Garay, J.A. (eds.) CRYPTO. Lecture Notes in Computer Science, vol. 8042, pp. 129–146. Springer (2013), <http://dx.doi.org/10.1007/978-3-642-40041-4>
40. 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), <http://dx.doi.org/10.1007/978-3-642-38348-9>
41. 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)
42. 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), <http://dx.doi.org/10.1007/978-3-642-03356-8>
43. Wang, X., Yin, Y.L., Yu, H.: Finding Collisions in the Full SHA-1. In: Shoup, V. (ed.) CRYPTO. Lecture Notes in Computer Science, vol. 3621, pp. 17–36. Springer (2005)
44. Wang, X., Yu, H.: How to Break MD5 and Other Hash Functions. In: Cramer, R. (ed.) EUROCRYPT. Lecture Notes in Computer Science, vol. 3494, pp. 19–35. Springer (2005)
45. Wang, X., Yu, H., Yin, Y.L.: Efficient collision search attacks on SHA-0. In: Shoup, V. (ed.) CRYPTO. Lecture Notes in Computer Science, vol. 3621, pp. 1–16. Springer (2005)
46. Yajima, J., Iwasaki, T., Naito, Y., Sasaki, Y., Shimoyama, T., Kunihiro, N., Ohta, K.: A strict evaluation method on the number of conditions for the SHA-1 collision search. In: Abe, M., Gligor, V.D. (eds.) ASIACCS. pp. 10–20. ACM (2008)