Freestart collision for full SHA-1

Marc Stevens[1⋆], Pierre Karpman[2,3⋆⋆], and Thomas Peyrin[3⋆⋆⋆]

[1] Centrum Wiskunde & Informatica, The Netherlands
[2] Inria, France
[3] Nanyang Technological University, Singapore

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

Abstract. We present in this article a freestart collision example 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, while only 10 days of computation on a 64 GPU cluster were necessary to perform the attack. This work builds on a continuous series of cryptanalytic advancements on SHA-1 since the theoretical collision attack breakthrough in 2005. In particular, we extend the recent freestart collision work on reduced-round SHA-1 from CRYPTO 2015 that leverages the computational power of graphic cards and adapt it to allow the use of boomerang speed-up techniques. We also leverage the cryptanalytic techniques by Stevens from EUROCRYPT 2013 to obtain optimal attack conditions, which required further refinements for this work.
Freestart collisions, like the one presented here, do not directly imply a collision for SHA-1. However, this work is an important milestone towards an actual SHA-1 collision and it further shows how graphics cards can be used very efficiently for these kind of attacks. 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/financial cost required by a SHA-1 collision computation. These projections are significantly lower than previously anticipated by the industry, due to the use of the more cost efficient graphics cards compared to regular CPUs. We therefore recommend the industry, in particular Internet browser vendors and Certification Authorities, to retract SHA-1 soon. We hope the industry has learned from the events surrounding the cryptanalytic breaks of MD5 and will retract SHA-1 before example signature forgeries appear in the near future. With our new cost projections in mind, we strongly and urgently recommend against a recent proposal to extend the issuance of SHA-1 certificates by a year in the CAB/forum (the vote closes on October 16 2015 after a discussion period ending on October 9).

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

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. They are crucial primitives for countless security applications, from digital signatures and Message Authentication Codes to password protection. One key security feature expected from a cryptographic hash function is collision resistance: it should be hard for an adversary to find two distinct messages M, $\hat{M}$ that hash to the same value $H(M) = H(\hat{M})$, where hard means with significantly less than $2^{\frac{n}{2}}$ computations.

A very famous hash function construction, followed by most industry standards, is the Merkle-Damgård paradigm [Mer89, Dam89]: 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. This construction is useful in particular for its simple security reduction: if the compression function is collision-resistant, then so is the corresponding hash function. However, one can distinguish between several types of collision resistances for the

⋆ 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)

compression function: a *freestart* collision is a pair of different message and chaining value (c, m), $(\hat{c}, \hat{m})$ such that $h(c, m) = h(\hat{c}, \hat{m})$, while a *semi-freestart* has the additional restriction that the chaining values c and $\hat{c}$ must be equal, i.e. $h(c, m) = h(c, \hat{m})$. It is very important to remark that the Merkle-Damgård security reduction assumes that any type of collision (freestart or semi-freestart) must be intractable by the adversary. Therefore, any type of collision break on the inner compression function should be taken very seriously as it removes directly any security protection coming from the operating mode for the hash function.

The most famous hash function family, basis for most hash function industry standards, is undoubtedly the `MD-SHA` family that originated with `MD4` [Riv90] and continued with `MD5` [Riv92] (due to serious security weaknesses [dB91, Dob96] found on `MD4` soon after its publication). Even though collision attacks on the compression function were quickly identified [dB93], the industry widely deployed `MD5` in applications where hash functions were required. Yet, in 2005, a team of researchers led by Wang [WY05] completely broke `MD5` and real collisions could even be computed for the entire hash function. This groundbreaking work inspired many further researches on this cryptanalysis direction. Even more worrying, Stevens *et al.* [SLdW07] showed that a more powerful type of collision attack so-called *chosen-prefix collision attack* could be performed against `MD5` and later, based on the inherent weaknesses of `MD5`, Stevens *et al.* [SSA+09] managed to forge a Rogue Certification Authority that in principle completely undermined HTTPS security. This is yet another argument for a very careful treatment of collision cryptanalysis advances: 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* [Ste13a] could be used to mitigate some of the risks during the migration).

While lessons should be learned from the `MD5` case, it is interesting to observe that the industry is again facing today a similar challenge. `SHA-1` [NIS95], designed by the NSA and a NIST standard, currently the most famous and probably most used hash function as of today, is facing important attacks since 2005. Based on previous successful cryptanalysis works [CJ98, BC04, BCJ+05] on `SHA-0` [NIS93] (`SHA-1`'s ancestor that only differs by a single rotation in the message expansion function), a team led again by Wang *et al.* [WYY05a] showed in 2005 the very first theoretical collision attack on `SHA-1`. This attack, while groundbreaking, remained theoretical for non-governmental entities as the 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 [DR06] (with a cost of 2^{35} `SHA-1` calls), 70 steps [DMR07] (cost 2^{44} `SHA-1`), 73 steps [Gre10] (cost $2^{50.7}$ `SHA-1`) and the latest advances for the hash function reached 75 steps [GA11] (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 state-of-the-art collision attack on full `SHA-1` was presented by Stevens [Ste13b] with an estimated cost of 2^{61} calls to the `SHA-1` compression function. Nevertheless, a publicly known collision still remained out of reach.

Very recently, `SHA-1` compression function collisions (more precisely freestart collisions) for a 76-round reduced version were obtained [KPS15] with a start-from-the-middle technique and a highly efficient GPU framework, requiring only a reasonable amount of GPU computation power (about 2^{50} `SHA-1`, which takes less than a week on a single card).

Because of these worrisome cryptanalysis advances on `SHA-1`, one is advised to use *e.g.* `SHA-2` [NIS02] or the new hash functions standard `SHA-3` [NIS15] instead when secure

hashing is needed. While NIST recommended that SHA-1-based certificates should not be trusted beyond 2014 [NIS12] (by 2010 for governmental use), the industry actors 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/financial cost required to generate a collision for SHA-1: in 2012, Bruce Schneier estimated (based on calculations from Jesse Walker) [Sch12] the SHA-1 collision attack cost 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.

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 [SSL15].

2 Our contributions.

In this paper, we exhibit for the very first time a collision for the full SHA-1 compression function.

More precisely, we present a freestart collision attack on the SHA-1 compression function for a computation cost approximately equivalent to 2^{57} SHA-1 evaluations.

Our work extends the start-from-the-middle freestart attack for 76-steps in [KPS15] to 80-steps. We build further on the framework in [KPS15] to adapt collision attacks for NVidia graphic cards, that have very high performance/cost ratio, and now incorporate the auxiliary paths (or boomerangs) speed-up technique from Joux and Peyrin [JP07]. We also leverage the cryptanalytic techniques by Stevens [Ste13b] to obtain optimal attack conditions, which required further refinements for this work.

We emphasize that our freestart collision counts as the first practical break of all 80 steps of SHA-1's compression function.

Table 2-1. A freestart collision for SHA-1

	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
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
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

As previously explained, recommendations on the retracting of SHA-1 are based on estimations of the resources needed to find SHA-1 collisions. We represent these resources both in terms of number of days on existing clusters with a certain number of recent graphic cards (CPUs), as well as the cost of renting cheap equivalent computational power on Amazon EC2 representing an amortized cost of operation including ownership, power and maintenance. Our freestart collision attack can be done in about 9 to 10 days on a

cluster with 64 GPUs, or by renting GPU time on Amazon EC2 for about 2K US$.[4] Based on experimental data obtained in this new work and the 2013 state-of-the-art collision attack, we can project that a real SHA-1 collision will take between 49 and 78 days on a 512 GPU cluster. Renting the equivalent GPU time on EC2 will cost between 75K US$ and 120K US$ and will plausibly take at most a few months.

The impact of our work is therefore not only theoretical. Freestart collisions are collisions for SHA-1's compression function, that do not directly translate to collisions for SHA-1, but do directly undermine the security proof of SHA-1. They represent an important alarm signal that warns users to quickly move away from using this hash function. In particular, we believe that our work shows that industry's plan to move away from SHA-1 in 2017 might not be soon enough.

Outline. In Section 3, we provide our analysis and recommendations regarding the timeline of migration from SHA-1 to SHA-2 or SHA-3. In Section 4 we give a short description of the SHA-1 hash function and our notations. In Section 5, we explain the structure of our cryptanalysis and the various techniques used from a high level point of view, and we later provide in Section 6 all the details of our attack for the interested readers.

3 Recommendations

Our work allowed to generate a freestart collision for the full SHA-1, but a plain 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/financial cost to generate such a collision with latest cryptanalysis advances [Ste13b] (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.* [WYY05a]).

Schneier's projections [Sch12] on the cost of SHA-1 collisions in 2012 (on EC2: ≈700K US$ by 2015, ≈173K US$ by 2018 and ≈43K US$ by 2021) were based on (an early announcement of) [Ste13b]. 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 Jan 2017 —one year before Schneier estimated that a SHA-1 collision would be within the resources of criminal syndicates.

However, as remarked in [KPS15] and now further improved in this article thanks to the use of boomerang speed-up techniques [JP07], graphics cards are much faster for this type of attacks (compared to CPUs) and we now precisely estimate that a full SHA-1 collision will cost between 75,000 and 120,000 US$ renting Amazon EC2 cloud over a few months today, 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 (SHA-1's predecessor) allowed the creation of a rogue Certification Authority undermining the security of all secure websites [SSA+09]. Collisions on SHA-1 can result in signature forgeries, but do not directly undermine the security of the Internet at large. More advanced so-called chosen-prefix collisions [SSA+09] are significantly more

[4] 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.

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

At the time of submission we just noted that in an ironic turn of events, the CA/Browser Forum is currently (from Oct 2 – Oct 9, 2015) holding a discussion to decide whether to extend issuance of SHA-1 certificates through the end of 2016 [For15]. With our new cost projections in mind, we strongly and urgently recommend against this proposal.

4 Preliminaries

4.1 SHA-1

We start this section with a short description of the SHA-1 hash function. We refer to the NIST specification document [NIS95] 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 [Dam89, Mer89], 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 which is 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. The block cipher itself is an 80-step (4 rounds of 20 steps each) five-branch generalized Feistel network. Its 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 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 4-1 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 4-1. Boolean functions and constants of SHA-1

round	step i	$f_i(B, C, D)$	K_i
1	$0 \leq i < 20$	$f_{IF} = (B \wedge C) \oplus (\overline{B} \wedge D)$	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 16 32-bit words $M_0, \ldots, M_{15}$. These 16 words are then expanded linearly

into the 80 32-bit words W_i as follows:

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

Both of the step function and the message expansion can easily be inverted.

4.2 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).

Local collisions. A main component of these attacks is the concept of *local collision*, introduced by Chabaud and Joux in 1998 to attack `SHA-0`, `SHA-1`'s predecessor [CJ98]. The idea underlying a local collision is first to introduce a difference in one of the intermediate 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 $W_{i...i+(j-1)}$ in order to cancel any contribution of the difference in A_i in the computation of a new internal state $A_{i+1...i+j}$, 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, these were chosen according to a linearized model (over $\mathbf{F}_2^n$) of the step functions.

The main obstacle when using local collisions is that the attacker does not control all of the message words, as some are generated by the message expansion; an important observation from Chabaud and Joux was to show how to chain multiple local collisions along a *disturbance vector* 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 is itself a message that has been expanded with the linear message expansion relation, where every '1'-bit marks the start of a local collision.

As each local collision in the last three rounds contributes to the overall complexity, one should use disturbance vectors that are sparse over these rounds. Initially, mostly heuristic cost functions were used to evaluate disturbance vector candidates, like Hamming weight of the disturbance vector (e.g., [BC04, PRR05, RO05, MP05, JP05]), sum of bitconditions for each local collision independently (not allowing carries) (e.g., [WYY05b, YIN+08]), and the product of independent local collision probabilities (allowing carries) (e.g., [MPRR06, Man11]). Manuel [Man08, Man11] noticed that all interesting disturbance vectors, including all disturbance vectors used in attacks in the literature, belong to two main classes $\mathrm{I}(K, b)$ and $\mathrm{II}(K, b)$. Within each class all disturbance vectors are forward or backward shifts (controlled by K) and/or bitwise cyclic rotations of each other (controlled by b). We will use this categorization notation throughout this paper.

Manuel also showed that success probabilities of local collisions are not always independent, causing biases into the above mentioned heuristic cost functions. This was later resolved by Stevens using a technique called joint local-collision analysis (JLCA)[Ste13b, Ste12], which allows to analyze entire sets of differential paths over the last 3 rounds that conform to 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, as well as a minimal set of conditions that achieves that probability. Although a direct approach is clearly unfeasible with an exponentially growing amount of possible differential paths, by exploiting the large amount of redundancy between all these differential paths to a very large extent, this can be done practically.

Non-linear differential path. A major improvement to the original attack entirely based on local collisions, was brought by Wang, Yin and Yu in 2005 in the first theoretical collision attack on the full `SHA-1` [WYY05a]. Their idea was to use a non-linear model for the propagation of the differences in the first few steps of `SHA-1` (in the form of a "non-linear" differential path) before using the linear model of local collisions from Chabaud and Joux. Even though the probability of two random messages following the non-linear path is extremely low, the attacker can use the freedom in the first 16 message words $M_{0...15}$ to ensure that this happens with probability one. Starting with a non-linear path eventually allowed to create better attacks than what was possible by using a purely linear view of the function.

Construction of non-linear differential paths was initially done by hand by Wang, Yin and Yu. Efficient algorithmic construction of such differential paths was later made possible in 2006 by De Cannière and Rechberger, who proposed a guess-and-determine approach [DR06]. A different approach based on a meet-in-the-middle method was also proposed by Stevens *et al.* [Ste12, HC].

Accelerating techniques. One can derive explicit sufficient conditions on state and message bits for the differential path. Such sufficient conditions allow 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 verifying state differences, allowing to stop computations on "bad" message pairs more early.

Another 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` [BC04]. 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. 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* [JP07], which allow more efficient attacks. Notably, their approach reuses local collisions as the base of the auxiliary paths.

5 Attack overview

In this section we provide an overview of how our attack was constructed. At a high level our attack construction 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. This is more fully explained in Section 5.1.
2. *Finding optimal attack conditions*: Having selected a disturbance vector, as explained in Section 5.2, 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 round 1 path leads to the highest possible success probability over all steps, yet minimizes the number of conditions within this model.

3. *Finding and analyzing boomerangs and neutral bits*: To speed up the freestart collision attack, we exploit advanced message modification techniques in the form of generalized neutral bits and boomerangs. We sample partial solutions for the attack conditions which we use to determine candidate boomerangs and neutral bits. No boomerang or neutral bit may invalidate any of the attack conditions. We also use sampling to estimate the probability of the boomerang/neutral bit interacting with particular sufficient conditions in the forward and backward direction. Though we do not allow significant interaction in the backward direction, we use these probabilities to determine at which step the boomerang/neutral bit are used, as explained in Section 5.3.
4. *Base solution generation*: Before we can apply neutral bits and/or boomerangs, we first need to compute a partial solution over 16 consecutive steps that can be extended using neutral bits and boomerangs. We call such a solution a *base solution*, which consists of state words $A_{-3}, \ldots, A_{17}$ and message words $W_1, \ldots, W_{16}$. The cost for generating base solutions is relatively low compared to the overall attack cost, therefore it is not heavily optimized and handled on regular CPUs. This is further explained in Section 5.4.
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 graphics cards that are significantly more cost-efficient than CPUs, using the highly efficient framework introduced by Karpman, Peyrin and Stevens [KPS15]. More details are provided in Section 5.5.

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 adaptions and improvements used for this work. Below we will more closely describe our procedures to show how all these techniques come together for our attack.

5.1 Disturbance vector selection

One can compute the highest success probability over the linear part exactly using joint-local collision analysis [Ste13b]. By further using the improvements described in [KPS15], one can restrict carries for the steps where sufficient conditions will be used and obtain the sufficient conditions for those steps immediately.

The sufficient conditions counts for the beginning of round 2, and the associated highest success probability for the remaining steps provide insight into the attack complexity under different circumstances. In Table 5-1 we give our analysis results for various DVs, listing the $-\log_2$ of the success probability over steps $[24, 80)$ assuming that all sufficient conditions up to A_{24} have been satisfied, as well as 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 [KPS15] with the difference between the costs $c_{[24,80)}$ for the 76-step attack and the DVs in the table.

For a full collision attack on `SHA-1`, the obvious choice here would be II(51,0) given the results in Table 5-1. However, for a freestart attack we have more constraints, in particular the IV differences are fixed as they have to cancel with the differences in $(A_{80}, B_{80}, C_{80}, D_{80}, E_{80})$. This impacts the estimated runtime as follows. If there are sufficient conditions on A_0 then the estimated runtime in the last column should be multiplied by $2^{c_{23}}$, as the 76-step freestart attack not having sufficient conditions on A_0 could ignore step 0. Moreover, if those IV differences are not sparse enough or at the wrong place then more neutral bits and boomerangs are likely to interact badly with the sufficient conditions

Table 5-1. Disturbance vector analysis. For each DV, under $c_{[24,80)}$, we list the $-\log_2$ 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

on the IV in the backwards direction. With not enough neutral bits and/or boomerangs the attack cost will be significantly higher.

Taking into account a preliminary analysis of available neutral bits and boomerangs, the best option appeared to be II(59,0). This choice actually leads to the same IV sufficient conditions that were used in the 76-step freestart attack with DV II(55,0).

5.2 Finding optimal attack conditions

As described above, using joint local collision analysis we obtain sufficient conditions for the beginning of round 2 and IV differences that are optimal (i.e., highest success probability). What remains is to construct a non-linear differential path for round 1. For this, we used the meet-in-the-middle method using the public HashClash implementation [HC]. Though we tried both non-linear differential path construction methods, *i.e.*, the guess-and-determine method 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 and that furthermore we could better position the conditions.

This round 1 differential path was used as input for another run of joint local collision analysis. Though in this case over all 80 steps replacing the assumed differences according to the disturbance vector with the differences in the A_i from the non-linear round 1 differential path, where changes in sign were allowed for sparse differences. In this manner joint local collision analysis is able to provide a complete set of attack conditions (*i.e.*, sufficient conditions for A_i and linear relations on message bits) that is optimized for highest success probability over the last three rounds, yet minimizing the amount of conditions needed.

JLCA will in fact spit out many complete sets that only vary slightly in signs. In fact, for our selected disturbance vector II(59,0) it turned out that a direct approach is far too costly and far too memory consuming, as the amount of complete sets grows exponentially with the number of steps we desired sufficient conditions for. 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 A_i-s 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 practicable.

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 quite some manual interaction.

The resulting sufficient conditions on the state are given in Figure 6-1 and the message bit relations are given in Figure 6-3.

5.3 Finding and analyzing neutral bits and boomerangs

Analyzing the boomerangs and neutral bits was done entirely automatically as described below. This process depends on a parameter called the main block offset that determines the offset of the message freedom window used during the attack. We have selected main block offset 5 as this led to the best distribution of usable neutral bits and boomerangs. This means that all neutral bits and boomerangs make changes to steps 5 up to 20, that propagate to step 4 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, we use more than 4000 sampled solutions for the given attack conditions over steps 1 up to 16. These 16 steps fully determine the message block, and also includes

the IV sufficient conditions and the dense round 1 non-linear differential path. For this step it is important to start over for every sample. If one uses message modification techniques to produce many samples then this would result in a very biased distribution where many samples would only differ in the last few steps.

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. Let $t \in [6, 16], b \in [0, 31]$ such that 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 $W_5, \ldots, W_{20}$. We verify that the boomerang is usable by checking that flipping the message 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 behavior of the boomerang over the backwards steps. For every sampled solution, we simulate the application of the boomerang by flipping the boomerang's active message. 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 behavior of the boomerang over the forward steps. For every sampled solution, we simulate the application of the boomerang by flipping the boomerang's active message. We then recompute steps 21 up to 79 forwards and keep track of any sufficient conditions that is broken on these steps. A boomerang will be used at step i in our attack if with probability at least 0.9 it will not break any sufficient condition up to step $i - 1$.

The neutral bit analysis procedure uses the same procedure to analyze the backward and forward steps, with the following changes. After boomerangs are determined, their conditions are added to the attack conditions and used to generate a new set of solution samples. Usable neutral bits consist of a set of 1 or more message bits that are flipped simultaneously. Let $t \in [5, 20], b \in [0, 31]$, flipping $W_t[b]$ may possibly break certain message bit relations. We express each message bit relation over $W_5, \ldots, W_{20}$ using linear algebra, and use Gaussian elimination to ensure that each message bit relation has a unique last message bit $W_i[j]$ (*i.e.* where $i*32+j$ is maximal). For each message bit relation involving $W_t[b]$, let $W_i[j]$ be the last message bit involved in the message bit relation. If (i, j) equals (t, b) then this neutral bit is not usable. Otherwise we add bit $W_i[j]$ as to be flipped together with $W_t[b]$ as part of the neutral bit. As for 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 similarly as described for boomerangs above.

The boomerangs we have selected are given in Figure 6-9 and the neutral bits are listed in Figure 6-6.

5.4 Base solution generation on CPU

The auxiliary conditions for boomerangs and neutral bits are all added to the attack conditions. Before we can apply neutral bits and/or boomerangs, 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 $1, \ldots, 16$ that fulfill all state conditions on $A_{-4}, \ldots, A_{17}$ and all message bit relations. A base solution itself

consists of state words $A_{-3}, \ldots, A_{17}$ and message words $W_1, \ldots, W_{16}$, but for the next step on the GPU, the message words are translated to the equivalent message words $W_5, \ldots, W_{20}$ with the main block offset of 5. The `C++` code for generation base solutions is automatically generated based on the given attack conditions and auxiliary conditions. In this manner, all intermediate steps and all tables can be hard-coded, and we can apply some static local optimizations eliminating unnecessary computations where possible. Though we do not exploit more advanced message modification techniques within these first 16 steps yet.

The base solution generation is only a small part of the overall complexity and is run entirely on the 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 a small factor more to ensure all GPUs have enough work.

5.5 Applying neutral bits and boomerangs on GPU

This is computationally the most cost intensive part, which is why we made use of more cost-efficient graphics cards for this part. In particular, we used 65 recent Nvidia GTX970 [NVI] graphics cards that feature 1664 small cores operating at a clock speed of about 1.2GHz; each card costs about 350 USD.[5] In [KPS15], the authors mention that a single GTX970 can be worth 322 CPU cores[6] cores for raw `SHA-1` operations, and about 140 CPU cores for their `SHA-1` attack.

We make use of the same efficient framework for Nvidia graphics cards [KPS15]. This makes use of the CUDA toolkit that provides programming extensions to `C` and `C++` for convenient programming. So 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 for that step; verify the sufficient conditions; and finally save the resulting extended solutions in the next global cyclic buffer for the next step. The smallest unit that can act independently on Nvidia graphics cards 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.

The exact details of which neutral bits and which boomerangs are used for each step are given in Section 6.

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, where there are no active disturbances in the disturbance vector. This is still done on the GPU. Solutions up to A_{60} are passed back to the CPU for further verification whether 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 is a serious issue.

[5] With the right motherboard one can place up to 4 such graphics cards in a single machine.
[6] Intel Haswell Core-i5 3.2GHz CPU.

6 Attack details

6.1 Sufficient conditions

We give a graphical representation of the differential path used in our attack up to step 28 in Figure 6-1, consisting of sufficient conditions for the state, and the associated message signed bit differences. The meaning of the bitcondition symbols are defined in Figure 6-2. Note that the signs of message bit differences are enforced through message bit relations. All message bit relations, including those for later steps, used in our attack are given in Figure 6-3 through Figure 6-5. 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. 6-1. The differential path used in the attack up to step 28.

6.2 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
- 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

symbol	condition on $(\mathbf{A_t[i]}, \mathbf{A'_t[i]})$
`.`	$A_t[i] = A'_t[i]$
`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. 6-2. Bitconditions

We give a graphical representation of the position of these neutral bits in Figure 6-6.

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]

We also give a graphical representation of this repartition in Figure 6-7.

Finally, we show how the neutral bits are packed together with the index of an (extended) base solution in Figure 6-8. Note that neutral bits on W_{17} are split between steps 18–20 and 21–25. Furthermore, the packing of steps 21-25 also includes some flip values.

6.3 The boomerangs

We give here the boomerangs used in the attack. One set is used with a first difference in the message initially introduced at W_{10}, to be used as neutral bits for step A_{28}; the second set uses a first difference at W_{11}, to be used as neutral bits for step A_{30}. There are 4 boomerangs in total, with initial difference at bits 7,8 of W_{10} and 8,9 of W_{11}. In Figure 6-9, we give a graphical representation of the complete set of message bits to be flipped for each boomerang. It is easy to see that these follow the pattern of a local collision.

```
- 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. 6-3. The message bit-relations used in the attack.

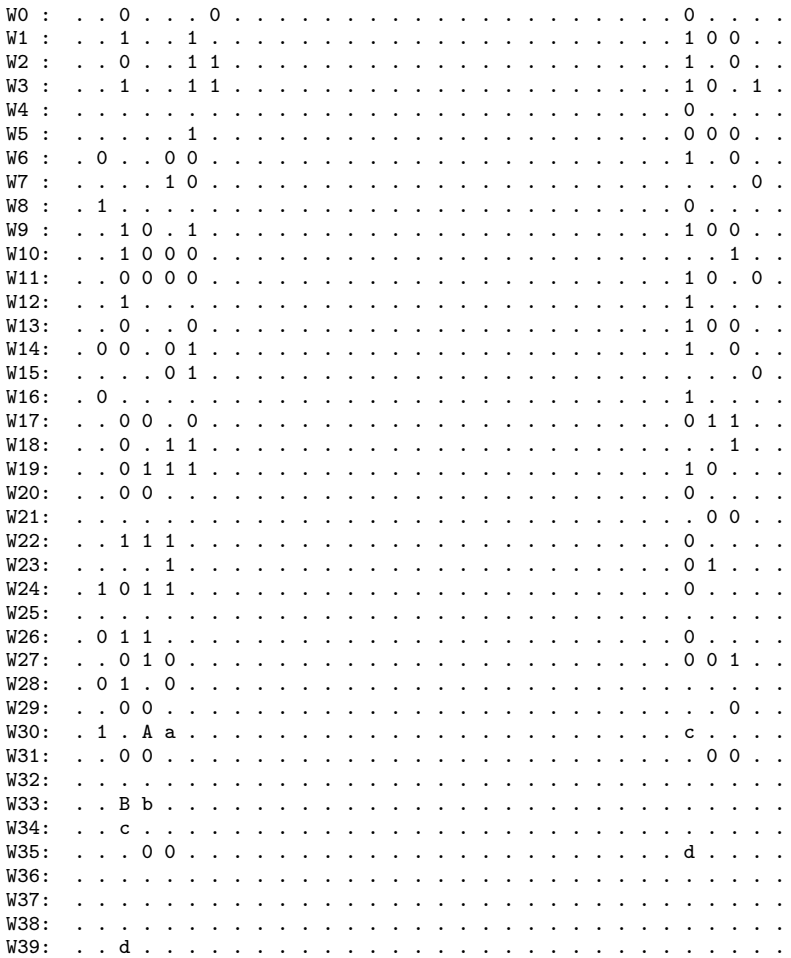

Fig. 6-4. The message bit-relations used in the attack for words W_0 to W_{39} (graphical representation). A "0" or "1" character represents a bit unconditionally set to 0 or 1. A pair of two letters x means that the two bits have the same value. A pair of two letters x and X means that the two bits have different values.

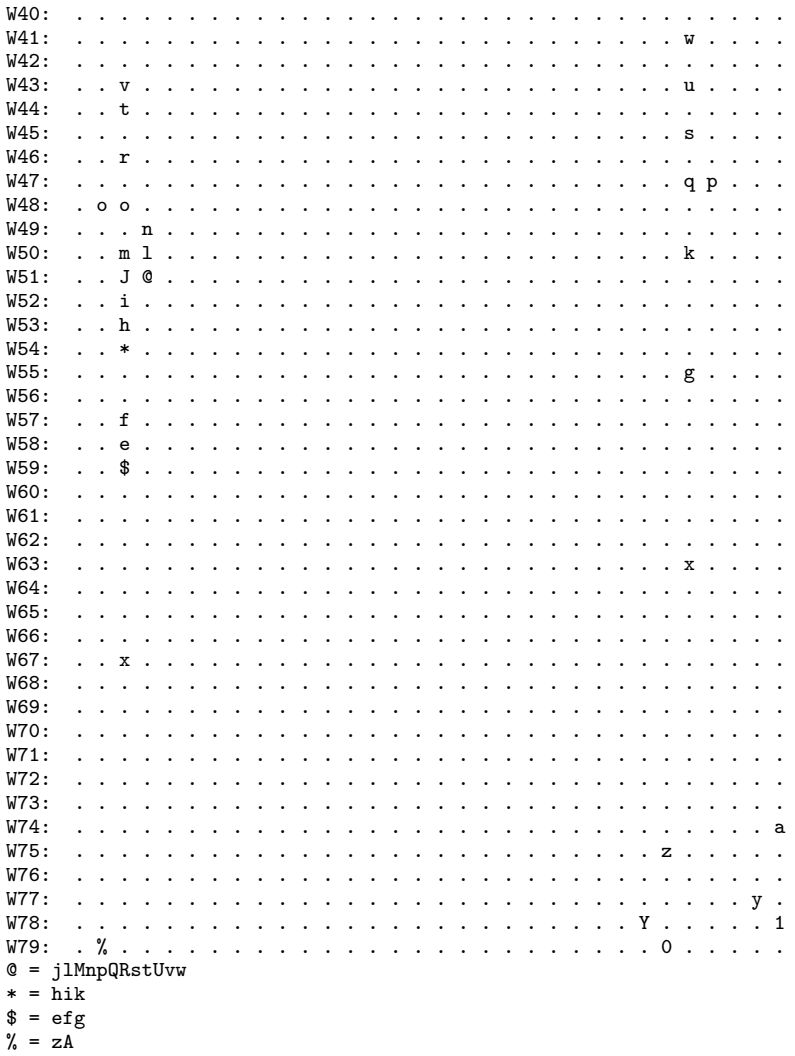

Fig. 6-5. The message bit-relations used in the attack for words $W_4 0$ to $W_7 9$ (graphical representation, continued). Non-alphanumeric symbols are used as shorthand for bit positions with more than one relation.

```
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. 6-6. 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.

```
A18:
W14 ........ ........ ....xxxx ........ (4 bits)
W15 ........ .......x xxx..... ........ (4 bits)
A19:
W14 ........ ........ ........ x.x..... (2 bits)
W15 ........ ........ ...xxxxx ........ (5 bits)
W16 ........ .......x xxxx.... ........ (5 bits)
A20:
W15 ........ ........ ........ x..x.... (2 bits)
W16 ........ ........ ....xxxx ........ (4 bits)
W17 ........ ....xxxx xx...... ........ (6 bits)
A21:
W17 ........ ........ ..xxxx.. ........ (4 bits)
W18 ........ ........ x....... ........ (1 bits)
A22:
W18 ........ ........ .xxxxxx. ........ (6 bits)
W19 ........ ........ .x...x.. ........ (2 bits)
A23:
W18 ........ ........ .......x xx.x.... (4 bits)
W19 ........ ........ ...xx.x. ........ (3 bits)
W20 ........ ........ x....... ........ (1 bits)
A24:
W19 ........ ........ .......x xx...... (3 bits)
W20 ........ ........ ..xxx... ........ (3 bits)
A25:
W20 ........ ........ ........ .x...... (1 bit)
```

Fig. 6-7. The 60 neutral bits regrouped by the first state where they start to interact. An "`x`" represents the presence of a neutral bit, and a "`.`" the absence thereof. The LSB position is the rightmost one.

```
For steps A18--20
W14:a W15:b W16:c W17:d basesol:i
ccccccccc...bbbbbbbbbb..baaaaa.a
dddddd......iiiiiiiiiiiiiiiiiii
For steps A21--25
W17:a W18:b W19:c W20:d extsol:i
Additional flips:F
bbbbbbbbbb.b..F.d.ddd....d.aaaaF
F.Fc.ccccccciiiiiiiiiiiiiiiiii
```

Fig. 6-8. The packing of the neutral bits and the (extended) base solution index. A letter {`a,b,c,d`} represents a bit on a given word, an "`F`" a flip bit, an "`i`" a bit of an (extended) solution index, and a "`.`" an unused bit (only present to maintain a proper alignment).

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

Fig. 6-9. The local collision patterns for each of the 4 boomerangs. The first difference to be introduced is highlighted with a capital letter. Note that boomerang "A" uses one more difference compared with the other boomerangs.

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. 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

The authors would like to express their gratitude to Orr Dunkelman for the use of his cluster with NVidia Tesla K10 cards.

References

[BC04] Eli Biham and Rafi Chen, *Near-Collisions of SHA-0*, CRYPTO (Matthew K. Franklin, ed.), Lecture Notes in Computer Science, vol. 3152, Springer, 2004, pp. 290–305.

[BCJ+05] Eli Biham, Rafi Chen, Antoine Joux, Patrick Carribault, Christophe Lemuet, and William Jalby, *Collisions of SHA-0 and Reduced SHA-1*, in Cramer [Cra05], pp. 36–57.

[Bra90] Gilles Brassard (ed.), *CRYPTO*, Lecture Notes in Computer Science, vol. 435, Springer, 1990.

[CJ98] Florent Chabaud and Antoine Joux, *Differential Collisions in SHA-0*, CRYPTO (Hugo Krawczyk, ed.), Lecture Notes in Computer Science, vol. 1462, Springer, 1998, pp. 56–71.

[Cra05] Ronald Cramer (ed.), *EUROCRYPT*, Lecture Notes in Computer Science, vol. 3494, Springer, 2005.

[Dam89] Ivan Damgård, *A Design Principle for Hash Functions*, in Brassard [Bra90], pp. 416–427.

[dB91] Bert den Boer and Antoon Bosselaers, *An Attack on the Last Two Rounds of MD4*, CRYPTO (Joan Feigenbaum, ed.), Lecture Notes in Computer Science, vol. 576, Springer, 1991, pp. 194–203.

[dB93] Bert den Boer and Antoon Bosselaers, *Collisions for the Compression Function of MD5*, EUROCRYPT (Tor Helleseth, ed.), Lecture Notes in Computer Science, vol. 765, Springer, 1993, pp. 293–304.

[DMR07] Christophe De Cannière, Florian Mendel, and Christian Rechberger, *Collisions for 70-Step SHA-1: On the Full Cost of Collision Search*, SAC (Carlisle M. Adams, Ali Miri, and Michael J. Wiener, eds.), Lecture Notes in Computer Science, vol. 4876, Springer, 2007, pp. 56–73.

[Dob96] Hans Dobbertin, *Cryptanalysis of MD4*, FSE (Dieter Gollmann, ed.), Lecture Notes in Computer Science, vol. 1039, Springer, 1996, pp. 53–69.

[DR06] Christophe De Cannière and Christian Rechberger, *Finding SHA-1 Characteristics: General Results and Applications*, ASIACRYPT (Xuejia Lai and Kefei Chen, eds.), Lecture Notes in Computer Science, vol. 4284, Springer, 2006, pp. 1–20.

[For15] CA/Browser Forum, *Ballot 152 - Issuance of SHA-1 certificates through 2016*, Cabforum mailing list, 2015.

[GA11] Evgeny A. Grechnikov and Andrew V. Adinetz, *Collision for 75-step SHA-1: Intensive Parallelization with GPU*, IACR Cryptology ePrint Archive **2011** (2011), 641.

[Gre10] Evgeny A. Grechnikov, *Collisions for 72-step and 73-step SHA-1: Improvements in the Method of Characteristics*, IACR Cryptology ePrint Archive **2010** (2010), 413.

[HC] *HashClash project webpage*, `https://marc-stevens.nl/p/hashclash/`.

[JP05] Charanjit S. Jutla and Anindya C. Patthak, *A Matching Lower Bound on the Minimum Weight of SHA-1 Expansion Code*, Cryptology ePrint Archive, Report 2005/266, 2005.

[JP07] Antoine Joux and Thomas Peyrin, *Hash Functions and the (Amplified) Boomerang Attack*, CRYPTO (Alfred Menezes, ed.), Lecture Notes in Computer Science, vol. 4622, Springer, 2007, pp. 244–263.

[KPS15] Pierre Karpman, Thomas Peyrin, and Marc Stevens, *Practical Free-Start Collision Attacks on 76-step SHA-1*, CRYPTO (Rosario Gennaro and Matthew Robshaw, eds.), Lecture Notes in Computer Science, vol. 9215, Springer, 2015, pp. 623–642.

[Man08] Stephane Manuel, *Classification and Generation of Disturbance Vectors for Collision Attacks against SHA-1*, Cryptology ePrint Archive, Report 2008/469, 2008.

[Man11] Stéphane Manuel, *Classification and generation of disturbance vectors for collision attacks against SHA-1*, Des. Codes Cryptography **59** (2011), no. 1-3, 247–263.

[Mer89] Ralph C. Merkle, *One Way Hash Functions and DES*, in Brassard [Bra90], pp. 428–446.

[MP05] Krystian Matusiewicz and Josef Pieprzyk, *Finding Good Differential Patterns for Attacks on SHA-1*, Coding and Cryptography, International Workshop, WCC 2005 (Øyvind Ytrehus, ed.), Lecture Notes in Computer Science, vol. 3969, Springer, 2005, pp. 164–177.

[MPRR06] Florian Mendel, Norbert Pramstaller, Christian Rechberger, and Vincent Rijmen, *The Impact of Carries on the Complexity of Collision Attacks on SHA-1*, FSE (Matthew J. B. Robshaw, ed.), Lecture Notes in Computer Science, vol. 4047, Springer, 2006, pp. 278–292.

[NIS93] National Institute of Standards and Technology, *FIPS 180: Secure Hash Standard*, May 1993.

[NIS95] National Institute of Standards and Technology, *FIPS 180-1: Secure Hash Standard*, April 1995.

[NIS02] National Institute of Standards and Technology, *FIPS 180-2: Secure Hash Standard*, August 2002.

[NIS12] National Institute of Standards and Technology, *Special Publication 800-57 - Recommendation for Key Management Part 1: General (Revision 3)*, July 2012.

[NIS15] National Institute of Standards and Technology, *FIPS 202: SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions*, August 2015.

[NVI] Nvidia Corporation, *Nvidia Geforce GTX 970 Specifications*, `http://www.geforce.com/hardware/desktop-gpus/geforce-gtx-970/specifications`.

[PRR05] Norbert Pramstaller, Christian Rechberger, and Vincent Rijmen, *Exploiting Coding Theory for Collision Attacks on SHA-1*, Cryptography and Coding, 10th IMA International Conference (Nigel P. Smart, ed.), Lecture Notes in Computer Science, vol. 3796, Springer, 2005, pp. 78–95.

[Riv90] Ronald L. Rivest, *The MD4 Message Digest Algorithm*, CRYPTO (Alfred Menezes and Scott A. Vanstone, eds.), Lecture Notes in Computer Science, vol. 537, Springer, 1990, pp. 303–311.

[Riv92] Ronald L. Rivest, *RFC 1321: The MD5 Message-Digest Algorithm*, April 1992.

[RO05] Vincent Rijmen and Elisabeth Oswald, *Update on SHA-1*, CT-RSA (Alfred Menezes, ed.), Lecture Notes in Computer Science, vol. 3376, Springer, 2005, pp. 58–71.

[Sch12] Bruce Schneier, *When Will We See Collisions for SHA-1?*, Schneier on Security, 2012.

[Sho05] Victor Shoup (ed.), *Advances in Cryptology — CRYPTO 2005: 25th Annual International Cryptology Conference, Santa Barbara, California, USA, August 14-18, 2005, Proceedings*, Lecture Notes in Computer Science, vol. 3621, Springer, 2005.

[SLdW07] Marc Stevens, Arjen K. Lenstra, and Benne de Weger, *Chosen-Prefix Collisions for MD5 and Colliding X.509 Certificates for Different Identities*, EUROCRYPT (Moni Naor, ed.), Lecture Notes in Computer Science, vol. 4515, Springer, 2007, pp. 1–22.

[SSA+09] Marc Stevens, Alexander Sotirov, Jacob Appelbaum, Arjen K. Lenstra, David Molnar, Dag Arne Osvik, and Benne de Weger, *Short Chosen-Prefix Collisions for MD5 and the Creation of a Rogue CA Certificate*, CRYPTO (Shai Halevi, ed.), Lecture Notes in Computer Science, vol. 5677, Springer, 2009, pp. 55–69.

[SSL15] *Survey of the SSL Implementation of the Most Popular Web Sites*, TIM Trustworthy Internet Movement, 2015.

[Ste12] Marc Stevens, *Attacks on Hash Functions and Applications*, Ph.D. thesis, Leiden University, June 2012.

[Ste13a] Marc Stevens, *Counter-Cryptanalysis*, CRYPTO (Ran Canetti and Juan A. Garay, eds.), Lecture Notes in Computer Science, vol. 8042, Springer, 2013, pp. 129–146.

[Ste13b] Marc Stevens, *New Collision Attacks on SHA-1 Based on Optimal Joint Local-Collision Analysis*, EUROCRYPT (Thomas Johansson and Phong Q. Nguyen, eds.), Lecture Notes in Computer Science, vol. 7881, Springer, 2013, pp. 245–261.

[WY05] Xiaoyun Wang and Hongbo Yu, *How to Break MD5 and Other Hash Functions*, in Cramer [Cra05], pp. 19–35.

[WYY05a] Xiaoyun Wang, Yiqun Lisa Yin, and Hongbo Yu, *Finding Collisions in the Full SHA-1*, in Shoup [Sho05], pp. 17–36.

[WYY05b] Xiaoyun Wang, Hongbo Yu, and Yiqun Lisa Yin, *Efficient Collision Search Attacks on SHA-0*, in Shoup [Sho05], pp. 1–16.

[YIN+08] Jun Yajima, Terutoshi Iwasaki, Yusuke Naito, Yu Sasaki, Takeshi Shimoyama, Noboru Kunihiro, and Kazuo Ohta, *A strict evaluation method on the number of conditions for the SHA-1 collision search*, ASIACCS (Masayuki Abe and Virgil D. Gligor, eds.), ACM, 2008, pp. 10–20.