SHA-1 is a Shambles: First Chosen-Prefix Collision on SHA-1 and Application to the PGP Web of Trust

Gaëtan Leurent, *Inria, France;* Thomas Peyrin,
Nanyang Technological University, Singapore

https://www.usenix.org/conference/usenixsecurity20/presentation/leurent

This paper is included in the Proceedings of the
29th USENIX Security Symposium.

August 12–14, 2020

978-1-939133-17-5

Open access to the Proceedings of the
29th USENIX Security Symposium
is sponsored by USENIX.

SHA-1 is a Shambles: First Chosen-Prefix Collision on SHA-1 and Application to the PGP Web of Trust*

Gaëtan Leurent
Inria, France
gaetan.leurent@inria.fr

Thomas Peyrin
Nanyang Technological University, Singapore
thomas.peyrin@ntu.edu.sg

Abstract

The SHA-1 hash function was designed in 1995 and has been widely used during two decades. A theoretical collision attack was first proposed in 2004 [29], but due to its high complexity it was only implemented in practice in 2017, using a large GPU cluster [23]. More recently, an almost practical *chosen-prefix* collision attack against SHA-1 has been proposed [12]. This more powerful attack allows to build colliding messages with two arbitrary prefixes, which is much more threatening for real protocols.

In this paper, we report the first practical implementation of this attack, and its impact on real-world security with a PGP/GnuPG impersonation attack. We managed to significantly reduce the complexity of collision attacks against SHA-1: on an Nvidia GTX 970, identical-prefix collisions can now be computed with a complexity (expressed in terms of SHA-1 equivalents on this GPU) of $2^{61.2}$ rather than $2^{64.7}$, and chosen-prefix collisions with a complexity of $2^{63.4}$ rather than $2^{67.1}$. When renting cheap GPUs, this translates to a cost of US$ 11k for a collision, and US$ 45k for a chosen-prefix collision, within the means of academic researchers. Our actual attack required two months of computations using 900 Nvidia GTX 1060 GPUs.

Therefore, the same attacks that have been practical on MD5 since 2009 are now practical on SHA-1. In particular, chosen-prefix collisions can break signature schemes and handshake security in secure channel protocols (TLS, SSH), if generated extremely quickly. We strongly advise to remove SHA-1 from those type of applications as soon as possible.

We exemplify our cryptanalysis by creating a pair of PGP/GnuPG keys with different identities, but colliding SHA-1 certificates. A SHA-1 certification of the first key can therefore be transferred to the second key, leading to an impersonation attack. This proves that SHA-1 signatures now offer virtually no security in practice. The legacy branch of GnuPG still uses SHA-1 by default for identity certifications, but after notifying the authors, the modern branch now rejects SHA-1 signatures (the issue is tracked as CVE-2019-14855).

*https://sha-mbles.github.io/

1 Introduction

Cryptographic hash functions are present in countless security applications and protocols, used for various purposes such as building digital signature schemes, message authentication codes or password hashing functions. In the key application of digital signatures for example, hash functions are classically applied on the message before signing it, as a domain extender and also to provide security guarantees. Informally, a cryptographic hash function H is a function that maps an arbitrarily long message M to a fixed-length hash value (we denote n its bit size). *Collision resistance* is the main security property expected from a hash function: it should be hard for an adversary to compute a collision (or *identical-prefix* collision), i.e. two distinct messages M and M' that map to the same hash value $H(M) = H(M')$, where by "hard" one means not faster than the generic $2^{n/2}$ computations birthday attack.

A cryptanalyst will try to find a collision for the hash function at a reduced cost, but ad-hoc collision attacks are hard to exploit in practice, because the attacker usually has little control over the value of the actual colliding messages (in particular where the differences are inserted, which are the interesting parts when attacking a digital signature scheme). Thus, one can consider stronger variants of the collision attack more relevant in practice, such as the so-called *chosen-prefix* collision [25] or CP collision. Two message prefixes P and P' are first given as challenge to the adversary, and his goal is to compute two messages M and M' such that $H(P \| M) = H(P' \| M')$, where $\|$ denotes concatenation. With such ability, the attacker can obtain a collision with arbitrarily chosen prefixes, potentially containing meaningful information.

A CP collision can also be found generically with $2^{n/2}$ computations (thus 2^{80} for a 160-bit hash function like SHA-1), but ad-hoc CP collision attacks are much more difficult to find than plain collision attacks, because of the uncontrolled internal differences created by the prefixes. Yet, a CP collision attack was found for the MD5 hash function [25], eventually leading to the creation of colliding X.509 certificates, and later of a rogue Certificate Authority (CA) [27]. CP collisions have also been shown to break important internet protocols, including TLS, IKE, and SSH [1], because they allow forgeries of the handshake messages if they can be generated extremely quickly.

Largely inspired by MD4 [19] and then MD5 [20], SHA-1 [16] is one the most famous cryptographic hash functions in the world, having been the NIST and de-facto worldwide hash function standard for nearly two decades. It remained a NIST standard until its deprecation in 2011 (and was forbidden for digital signatures at the end of 2013). Indeed, even though its successors SHA-2 or SHA-3 are believed to be secure, SHA-1 has been broken by a theoretical collision attack in 2004 [29]. Due to its high technicality and computational complexity (originally estimated to about 2^{69} hash function calls), this attack was only implemented in practice in 2017, using a large GPU cluster [23]. Unfortunately, the SHA-1 deprecation process has been quite slow and one can still observe many uses of SHA-1 in the wild, because it took more than a decade to compute an actual collision, plain collisions are difficult to use directly to attack a protocol, and migration is expensive.

Very recently, a CP collision attack against SHA-1 has been described in [12] (but not implemented), which requires an estimated complexity between $2^{66.9}$ and $2^{69.4}$ SHA-1 computations. It works with a two-phase strategy: given the challenge prefixes and the random differences on the internal state it will induce, the first part of the attack uses a birthday approach to limit the internal state differences to a not-too-big subset (as done in [22, 25]). From this subset, reusing basic principles of the various collision search advances on SHA-1, one slowly adds successive message blocks to come closer to a collision, eventually reaching the goal after a dozen blocks. Even though these advances put the CP collisions within practical reach for very well-funded entities, it remains very expensive to conduct and also very difficult to implement as the attack contains many very technical parts.

1.1 Our Contributions

In this article, we exhibit the very first chosen-prefix collision against SHA-1, with a direct application to PGP/GnuPG security. Our contributions are threefold.

Function	Collision type	Cost	Ref.
SHA-1	free-start collision	$2^{57.5}$	[24]
	collision	2^{69}	[29]
		$2^{64.7}$	[22, 23][a]
		$2^{61.2}$	New
	chosen-prefix collision	$2^{77.1}$	[22]
		$2^{67.1}$	[12]
		$2^{63.4}$	New

Table 1: Comparison of previous and new cryptanalysis results on SHA-1. A free-start collision is a collision of the compression function only, where the attacker has full control on all the primitive's inputs. Complexities in the table are given in terms of SHA-1 equivalents on a GTX-970 GPU (when possible).

[a] Equivalent to 2^{61} SHA-1 on CPU, $2^{64.7}$ on GPU

Complexity improvements. While the work of [12] was mostly about high-level techniques to turn a collision attack into a chosen-prefix collision attack, we have to look at the low-level details to actually implement the attack. This gave us a better understanding of the complexity of the attack, and we managed to significantly improve several parts of the attacks (See Table 1).

First, we improved the use of degrees of freedom (neutral bits [3] and boomerangs [10]) during the search for near-collision blocks. This reduces the computational complexity for both plain and chosen-prefix collision attacks, leading to important savings: on an Nvidia GTX 970, plain collisions can now be computed with a complexity of $2^{61.2}$ rather than $2^{64.7}$ (expressed in terms of SHA-1 equivalents on this GPU). We note that the general ideas underlying these improvements might be interesting for cryptanalysis of algorithms beyond SHA-1.

Second, we improved the graph-based technique of [12] to compute a chosen-prefix collision. Using a larger graph and more heuristic techniques, we can significantly reduce the complexity of a chosen-prefix collision attack, taking full advantage of the improvements on the near-collision block search. This results in a chosen-prefix collision attack with a complexity of $2^{63.4}$ rather than $2^{67.1}$.

Record computation. We implemented the entire chosen-prefix collision attack from [12], with those improvements. This attack is extremely technical, contains many details, various steps, and requires a lot of engineering work. Performing such a large-scale computation is still quite expensive, but is accessible with an academic budget. More precisely, we can can rent cheap GPUs from providers that use gaming or mining cards in consumer-grade PCs, rather that the datacenter-grade hardware used by big cloud providers. This gives a total

cost significantly smaller than US$ 100k to compute a chosen-prefix collision. We give more detailed complexity and cost estimates in Table 2.

We have successfully run the computation over a period of two months, using 900 GPUs (Nvidia GTX 1060). Our attack uses one partial block for the birthday stage, and 9 near-collision blocks. We paid US$ 75k to rent the GPUs from GPUserversrental, but the actual price could be smaller because we lost some time tuning the attack. There is also a large variability depending on luck, and GPU rental prices fluctuate with cryptocurrency prices.

PGP/GnuPG impersonation. Finally, in order to demonstrate the practical impact of chosen-prefix collisions, we used our CP collision for a PGP/GnuPG impersonation attack. The chosen prefixes correspond to headers of two PGP identity certificates with keys of different sizes, an RSA-8192 key and an RSA-6144 key. By exploiting properties of the OpenPGP and JPEG format, we can create two public keys (and their corresponding private keys): key A with the victim name, and key B with the attacker name and picture, such that the identity certificate containing the attacker key and picture leads to the same `SHA-1` hash as the identity certificate containing the victim key and name. Therefore, the attacker can request a signature of his key and picture from a third party (from the Web of Trust or from a CA) and transfer the signature to key A. The signature stays valid because of the collision, while the attacker controls key A with the name of the victim, and signed by the third party. Therefore, he can impersonate the victim and sign any document in her name.

1.2 `SHA-1` Usage and Impact

Our work shows that `SHA-1` is now fully and practically broken for use in digital signatures. GPU technology improvements and general computation cost decrease will further reduce the cost, making it affordable for any ill-intentioned attacker in the very near future.

`SHA-1` usage has significantly decreased in the last years; in particular web browsers now reject certificates signed with `SHA-1`. However, `SHA-1` signatures are still supported in a large number of applications. `SHA-1` is the default hash function used for certifying PGP keys in the legacy branch of GnuPG (v 1.4), and those signatures were accepted by the modern branch of GnuPG (v 2.2) before we reported our results. Many non-web TLS clients also accept `SHA-1` certificates, and `SHA-1` is still allowed for in-protocol signatures in TLS and SSH. Even if actual usage is low (a few percent), the fact that `SHA-1` is *allowed* threatens the security because a man-in-the-middle attacker can downgrade the connection to `SHA-1`. `SHA-1` is also the foundation of the GIT versioning system, and it is still in DNSSEC signatures. There are probably a lot of less known or proprietary protocols that still use `SHA-1`, but this is more difficult to evaluate.

1.3 Outline

We first recall `SHA-1` inner workings and previous cryptanalysis on this hash function in Section 2. We then provide improvements over the state-of-the-art `SHA-1` collision attacks in Section 3 and Section 4, and we describe the details of the `SHA-1` chosen-prefix collision computation in Section 5. Finally, we show a direct application of our CP collision attack with a PGP/GnuPG impersonation (together with discussions on other possible applications) in Section 6. We discuss `SHA-1` usage and the impact of our results in Section 7. Eventually, we conclude and propose future works in Section 8.

2 Preliminaries

In this section, we describe the `SHA-1` hash function (we refer to [16] for all the complete details) and summarize the previous cryptanalysis relevant to our new work.

2.1 Description of SHA-1

`SHA-1` is a 160-bit hash function that follows the well-known Merkle-Damgård paradigm [6, 15], with 512-bit message blocks, and a 160-bit state. The `SHA-1` compression function uses the Davies-Meyer construction, that turns a block cipher E into a compression function: $cv_{i+1} = E_{m_{i+1}}(cv_i) + cv_i$, where $E_k(y)$ is the encryption of the plaintext y with the key k, and $+$ is a word-wise 32-bit modular addition. It is composed of 4 rounds of 20 steps each (for a total of 80 steps), where one step follows a generalised Feistel network. Since only a single register value is updated, the other registers being only rotated copies, we can express the `SHA-1` step function using a single variable:

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

where K_i are predetermined constants and f_i are boolean functions (given in Table 3). For this reason, the differential trails figures in this article will only represent A_i, the other register values at a certain point of time can be deduced directly.

The extended message words W_i are computed linearly from the incoming 512-bit message block m, the process being called message extension. One first splits m into 16 32-bit words $M_0, \ldots, M_{15}$, and then the W_i's

Function	Collision type	GPU	Time	Complexity	Cost
SHA-1	collision	GTX 970	22 years	$2^{61.2}$	
		GTX 1060	27 years	$2^{61.6}$	US$ 11k
		GTX 1080 Ti	8 years	$2^{61.6}$	
	chosen-prefix	GTX 970	99 years	$2^{63.4}$	
		GTX 1060	107 years	$2^{63.5}$	US$ 45k
		GTX 1080 Ti	34 years	$2^{63.6}$	
MD5 ‖ SHA-1	both (plain or CP)	GTX 970	1400 years	$2^{67.2}$	
		GTX 1060	1700 years	$2^{67.6}$	US$ 720k
		GTX 1080 Ti	540 years	$2^{67.6}$	

Table 2: Complexity of the attacks against SHA-1 reported in this paper on several GPUs. The complexity is given in SHA-1 equivalents (using hashcat benchmarks). For the cost evaluation we assume that one GTX 1060 GPU can be rented for a price of US$ 35/month (the two phases of the attack are easily parallelisable): https://web.archive.org/web/20191229164814/https://www.gpuserversrental.com/
To attack MD5 ‖ SHA-1, we use the multicollision attack of Joux [9] with three phases: (i) a CP collision on SHA-1, (ii) 64 collisions on SHA-1, and (iii) 2^{64} evaluations of MD5.

step i	$f_i(B,C,D)$	K_i
$0 \le i < 20$	$(B \wedge C) \oplus (\overline{B} \wedge D)$	0x5a827999
$20 \le i < 40$	$B \oplus C \oplus D$	0x6ed6eba1
$40 \le i < 60$	$(B \wedge C) \oplus (B \wedge D) \oplus (C \wedge D)$	0x8fabbcdc
$60 \le i < 80$	$B \oplus C \oplus D$	0xca62c1d6

Table 3: Boolean functions and constants of SHA-1

are computed as follows:

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

In the rest of this article, we will use the notation $X[j]$ to refer to bit j of word X.

2.2 Previous Works

We recall here the general state-of-the-art collision search strategies that we will use for our CP collision attack. Readers only interested by the applications of our CP collision attack can skip up to Section 6. In the rest of the article, unless stated otherwise, difference will refer to the XOR difference between two bits or the bitwise XOR difference between two words.

2.2.1 Differential Trails

The first results on SHA-0 (predecessor of SHA-1) and SHA-1 were differential attacks using trails built by linearizing the compression function (we call these *linear paths*, in opposition to *non-linear paths* which have been built without linearization), assuming that modular additions and boolean functions f_i in the SHA-1 compression function are behaving as an XOR. More precisely, the

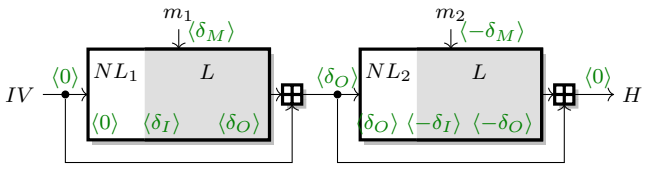

Figure 1: 2-block collision attack using a linear trail $\delta_I \overset{\delta_M}{\rightsquigarrow} \delta_O$ and two non-linear trails $0 \rightsquigarrow \delta_I$ and $\delta_O \rightsquigarrow -\delta_I$ in the first 10∼15 steps. Green values between bracket represent differences in the state.

32-bit modular addition is replaced by a 32-bit bitwise XOR and the f_i functions are replaced by 3-input XOR operations. The trails are generated with a succession of so-called local collisions: small message disturbances whose influence is immediately corrected with other message differences inserted in the subsequent SHA-1 steps, taking advantage of the linear message expansion.

In 2005, the seminal work of Wang *et al.* [29] showed that non-linear differential trails (trails generated without linearizing the SHA-1 step function) can be used for the first 10∼15 steps of the compression function, connecting any incoming input difference to any fixed difference δ_I at step 10∼15. Due to the Davies-Meyer construction used in SHA-1, this gives a collision attack with two successive blocks, using the same differential trail from step 10∼15 to 80 (just using opposite signs: $0 \overset{\delta_M}{\rightsquigarrow} \delta_O$ and $\delta_O \overset{-\delta_M}{\rightsquigarrow} -\delta_O$), as seen in Figure 1.

2.2.2 Improving the Efficiency of Collision Search

Once the differential trail is set the attacker must find a pair of messages that follows it. A simple strategy uses an

early-abort tree exploration for the 16 first steps, taking advantage of the degrees of freedom in the message, while the remaining steps are probabilistic. More advanced amortization methods (neutral bits [3], boomerangs [10, 11] or message modification [29]) are used to control more than 16 steps. Because of this amortization, usually the first 20 or so steps (which hold with a low probability because of the non-linear trail) do not contribute to the final complexity of the attack.

Neutral bits were first introduced for the cryptanalysis of `SHA-0` [2, 3]. The idea is to find a small message modification (one or a few bits), that does not interact with necessary conditions in the differential path before a certain step x. Once a message pair following the differential path until step x is found, one can get another pair valid until step x by applying the modification. The probability that a modification is neutral until a step x can be pre-analysed before running the attack. A key observation is that any combination of two of more neutral bits until step x is likely to also be neutral until step x.

Boomerangs [10] or tunnels [11] are very similar amortization tools to neutral bits. Basically, they can be seen as neutral bits that are planned in advance. A perturbation built from one or a few local collisions (or relaxed versions) is neutral to the differential path after a few steps with a certain probability, but extra conditions are forced in the internal state and message to increase this probability. Boomerangs are generally more powerful than neutral bits (they can reach later steps than classical neutral bits), but consume more degrees of freedom. For this reason, only a few of them can be used, but their amortization gain is almost a factor 2.

Note that a lot of details have to be taken into account when using neutral bits or boomerangs, as many equations between internal state bits and message bits must be fulfilled in order for the differential path to be valid. Thus, they can only be placed at very particular bit positions and steps.

2.2.3 Chosen-prefix Collision Attacks

Chosen-prefix collision attacks are much harder than identical-prefix attacks because they have to start from a random difference in the internal state. To alleviate this difficulty, the first chosen-prefix collision attack (demonstrated on `MD5` [25]) introduced a birthday search phase, processing random message blocks until the chaining variable difference δ belongs to a large predetermined set $\mathcal{S}$. The set $\mathcal{S}$ contains differences that can be slowly erased by a succession of near-collision blocks, eventually leading to a collision. Due to the birthday paradox, it is possible to reach a difference in $\mathcal{S}$ with birthday complexity $\sqrt{\pi \cdot 2^n/|\mathcal{S}|}$.

This two-phase strategy (see Figure 2) was first applied to `SHA-1` in [22], for a cost of $2^{77.1}$ hash calls. The improvement compared to the generic 2^{80} attack is not very large, due to the difficulty for an attacker to build a large set of differences that can be erased efficiently with a near-collision block. In [22] a set $\mathcal{S}$ of 192 allowable differences was used, corresponding to differences that can be reached with a single near-collision block, using a fixed differential trail, varying the signs of the differences, and letting some uncontrolled differences spread during the very last steps.

This was improved in [12] by increasing the size of the set $\mathcal{S}$. First the set of possible differences that can be reached efficiently with a near-collision block was increased to 8768 elements. Another crucial improvement from [12] is the use of a multi-block strategy for `SHA-1` that significantly increases the size of the set $\mathcal{S}$: it contains differences δ that can be decomposed as $\delta = -\left(\delta_O^{(1)} + \delta_O^{(2)} + \cdots + \delta_O^{(r)}\right)$, where each $\delta_O^{(i)}$ can be reached as the output of a differential trail. Therefore, the attacker just has to find near-collision blocks with output differences $\delta_O^{(1)}, \ldots, \delta_O^{(r)}$, where each near-collision block will cancel one of the differences $\delta_O^{(i)}$ composing δ. In particular, a clustering effect appears with this multi-block strategy, which can be leveraged by the attacker to select dynamically the allowable differences at the output of each successive block, to further reduce the attack complexity. This resulted in an estimated CP collision search complexity in the range of $2^{66.9}$ to $2^{69.4}$ hash evaluations, surprisingly not much greater than that of finding a simple collision.

3 Improving `SHA-1` Collision Attack

Our first contribution is an improvement of the collision attack from Eurocrypt 2013 [22] and its GPU implementation from Crypto 2017 [23]. Through better use of degrees of freedom (message modifications and boomerangs) and code improvements, we gained a factor between 8 and 10 (depending on GPU architecture) on the time needed to find a conforming block.

Since this part of our work is very technical, we only give an overview of our results in this section. Technical details can be found in the full version of the paper [13] and the corresponding code is available at https://github.com/SHA-mbles/sha1-cp.

3.1 Analysis of Previous Works

First, we observed some differences between the theoretical analysis of [22] and the practical implementation

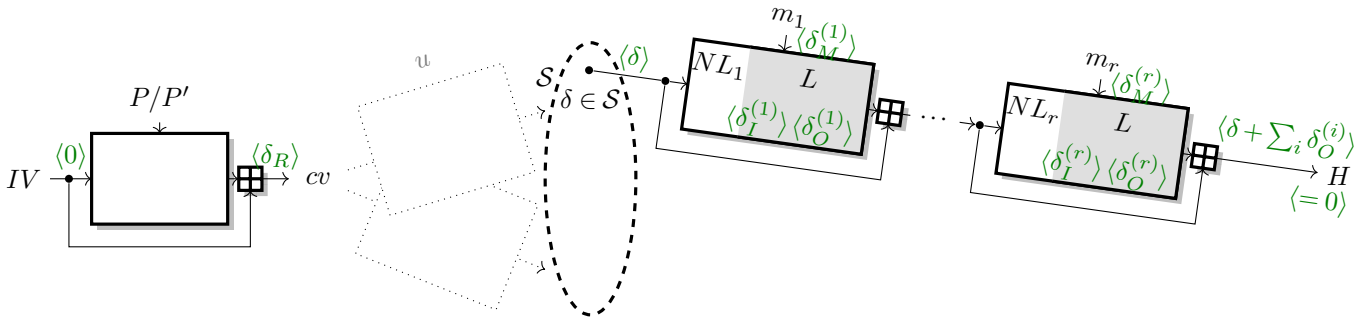

Figure 2: High-level view of a chosen-prefix collision attack. We assume that differences $\delta \in \mathcal{S}$ can be decomposed as $\delta = -\left(\delta_O^{(1)} + \delta_O^{(2)} + \cdots + \delta_O^{(r)}\right)$, where each $\delta_O^{(i)}$ can be reached as the output of a differential trail.

of [23]. One of the boomerangs (on bit 6 of M_6) contradicts one of the conditions used to maximize the probability of the path. Using this boomerang still improves the attack, because the gain in efficiency is larger that the loss in probability, but this affect the complexity evaluation. Similarly, one of the neutral bits used in the GPU code (on bit 11 of M_{13}) contradicts another condition in the differential path, leading to an increase in complexity of a factor $2^{0.2}$.

In our analysis, we assume that the neutral bit on bit 11 of M_{13} is not used, and that the boomerang on bit 6 of M_6 is only used for the last near-collision block, where the speed-up is most noticeable, and we have enough degrees of freedom to include all the boomerangs without difficulty. Therefore we can estimate more accurately the complexity of the previous CP attack [12] as $2^{67.1}$ `SHA-1` computations, instead of the range of $2^{66.9}$ to $2^{69.4}$ reported previously.

3.2 Additional Boomerangs

We found some additional boomerangs that can be used to speed-up the attack, on bits 4, 5, and 6 of M_{11}. Those boomerangs are not used in previous attacks because they interact badly with conditions of the differential trail, but this can be fixed by changing the last correction of the boomerangs to be a modular addition correction instead of an XOR correction.

More precisely, boomerangs are based on local collisions: an initial message difference introduces a difference in the state and another message difference cancels the state difference at a later step. In previous works, both message differences affect a single bit, so that they can be considered either as an XOR difference or as a modular difference. In this work, we only enforce a fixed modular difference for some boomerangs; depending on the value of the initial message, this difference will affect one or several bits (due to carries). Therefore, we can relax some of the conditions and make the additional boomerangs compatible with the differential path.

3.3 Precise Conditions of Neutral Bits

We also improved the rate of partial solutions generated by looking more precisely at the effect of each neutral bit. In particular, we found that some neutral bits flip with very high probability a certain condition after the step for which they are considered neutral. Therefore, these bits can be used as message modifications rather than neutral bits: instead of considering both the initial message and the message with the neutral bit applied and to test both of them at the later step, we can directly test the condition and decide which message to consider. Using this bit as message modification instead of neutral bit is more efficient, as one invalid branch in the search tree will be rightfully not explored.

In some cases, we also found that a bit that is neutral up to step i can only break some of the conditions of step i, while the rest will never be impacted. Therefore, we can test the conditions that are not affected before using that neutral bit, so as to avoid unnecessary computations. This strategy can be seen as a more precise neutral bit approach, where the attacker doesn't work step-wise, but instead condition-wise: more fine-grained filtering will lead to computation savings.

All in all, these tricks result in a better exploration of the collision search tree by cutting branches earlier. We give detailed benchmarks results and complexity estimates in Table 4, after implementing our improvements in the code of [23] (where an A_i-solution refers to an input pair that is following the differential path until word A_i inclusive).

3.4 Building Differential Trails

Following [12], we try to reuse as much as possible the previous works on `SHA-1`, and to keep our differential trail as close as possible to the attack of Stevens *et*

al. [23], out of simplicity. More precisely, for each block of the collision phase, as starting point we reused exactly the same core differential path as in [23]: the difference positions in the message are the same, and the difference positions in the internal state are the same after the first 13 steps (roughly). We also tried to keep difference signs to be the same as much as possible. However, we made some modifications to the boomerangs and neutral bits as explained in the previous subsection.

The starting path skeleton is depicted in Figure 3. For each new block of the near-collision phase, we:

1. collect the incoming chaining variable and its differences and insert them inside the skeleton;
2. set the signs of the differences in the very last steps (chosen so as to minimize the final collision complexity according to the graph, see Section 4) and generate the linear system of all equations regarding the message words;
3. compute a valid non-linear differential path for the first steps;
4. generate base solutions, *i.e.* partial solutions up to A_{14}, possibly using help of neutral bits;
5. from the base solutions, search for a pair of messages that fulfils the entire differential path, using neutral bits, message modifications and boomerangs.

Steps 1 to 4 are done on CPU because they are not too computationally intensive, but step 5 runs on GPU.

In comparison with a classical collision attack [23], our paths have fewer degrees of freedom because of additional constraints on the late-step message bits, and denser input difference on the chaining variable. However, we had enough degrees of freedom to find a conforming messages pair for all blocks during the attack. The use of the additional short boomerangs reduces also the number of neutral bits that can be used, but we still had enough to keep the GPU busy (in stage 5) while the CPU was producing the base solutions (in stage 4), even though our computation cluster is composed of low range CPUs.

4 Improving SHA-1 CP Collision Attack

In order to take advantage of the low-level improvements to collision attack techniques, we must also improve the high-level chosen-prefix collision attack.

The complexity of the birthday phase depends on the size of the set $\mathcal{S}$ of differences that can be erased from the state, therefore we need a larger set. For the near-collision phase, the complexity depends on how we combine the near-collision blocks to erase the difference in the state. We improve the graph techniques of [12] and suggest a more heuristic approach, resulting in a lower average complexity, but without a guaranteed upper bound.

4.1 Graph Construction

In order to efficiently erase the differences from the set $\mathcal{S}$, [12] uses a graph where vertices are the state difference in $\mathcal{S}$, and there is an edge between δ and δ' if $\delta' - \delta$ can be obtained as the output difference of the compression function (using a near-collision block). The birthday phase designates a starting node in the graph and we just have to follow a path leading to the zero difference, as illustrated in Figure 4. For each edge, we search for a block with the correct output difference, using near-collision search, with a cost that depends on the target difference. In the following, we denote the cost for the optimal output differences as C_{block}; it is equivalent to the cost of an identical-prefix collision.

Large graph. We started with the same approach as in [12], building a series of graphs with increasing limits on the number of blocks allowed. More precisely, we consider the set of all nodes that are reachable with a path of cost at most 24 C_{block} and up to 10 blocks. This results in a graph with $2^{36.2}$ nodes[1], which requires 2TB of storage (storing only the nodes and their cost).

Clustering. In order to minimize the complexity of the near-collision phase of the attack, [12] uses a clustering technique to exploit multiple paths in the graph (see Figure 5). Indeed, the near-collision search does not have to commit to a fixed output difference. When two output differences correspond to useful paths in the graph and are compatible with the same differential path, the attacker can run the near-collision search and stop as soon as one of them is obtained.

Concretely, let us assume we have two output differences δ_1 and δ_2 compatible with the same differential trail, that can each be reached with a cost of C_{block}. There are two different ways to erase a difference $-\delta_1 - \delta_2$ in the state:

- a block with difference δ_1, followed by a block with difference δ_2;
- a block with difference δ_2, followed by a block with difference δ_1.

If we don't decide in advance the target difference for the first block, the search is expected to reach either δ_1 or δ_2 with a cost of only 0.5 C_{block}, leading to an attack complexity of 1.5 C_{block} rather than 2 C_{block}.

In our case, we initially consider nodes at distance up to 24 C_{block} and we run the clustering technique to get a better estimate of the complexity when we don't specify in advance the sequence of differences. After several weeks of computation on a machine with 48 cores and

[1] The largest graph suggested in [12] has size $2^{33.7}$.

GPU	arch	Hashrate	Collision (old)		Collision (new)		
			A_{33} rate	SHA-1	A_{33} rate (r)	SHA-1	Gain
K20x (1 GPU)	Kepler	1.7GH/s	28k/s	$2^{64.4}$	255k/s	$2^{61.2}$	9.1
GTX 970	Maxwell	3.9GH/s	59k/s	$2^{64.5}$	570k/s	$2^{61.2}$	9.6
GTX 1060	Pascal	4.0GH/s	53k/s	$2^{64.7}$	470k/s	$2^{61.6}$	8.8
GTX 1080 Ti	Pascal	12.8GH/s	170k/s	$2^{64.7}$	1500k/s	$2^{61.6}$	8.8

Table 4: Cost of collision attacks. One collision requires on average $2^{48.5}$ A_{33}-solutions (those results include the boomerang on M_6[8]).
Note: we use the hashrate from hashcat, which is slightly over-optimistic (i.e. attack cost in SHA-1 computations is overestimated).

i	A_i	W_i
-4:	| |	
-3:	| |	
-2:	| Incoming Chaining Variable |	
-1:	| |	
00:	|_____|	----xx----------x-
01:	????????????????????????????????--	xx----------x----
02:	????????????????????????????????	x-xx-x----------xxx--
03:	????????????????????????????????	--xxxx----------x--
04:	????????????????????????????????	x-xxxx----------xx-x-
05:	????????????????????|?|????????	-x----------x----
06:	????????????????????|?|????????	--x--x----------0-0-xxx--
07:	????----------0-0??????	xxx-xx----------1-1-----x-x--
08:	???x----------|--0?0--??	----xx----------x-
09:	???----------|--1?1--??	xx----------0--x----
10:	???----------|0|?---??	x-xx-x----------1------xxx--
11:	??x----------|0|0-----	--x-xx----------111-x--
12:	----------111----	x-xxux----------xx---
13:	n----------000--	x-xx----------1u----
14:	--n----------111--	----------1-xx--
15:	u-1-1----------	x-xxx----------n----
16:	un0-0----------	----u----------nu---
17:	u--1----------	-xxnn----------n----
18:	u-u0----------	--0-n----------n-n--
19:	u----------	-xuu----------n----
20:	u-u----------	x-nux----------nnu--

Figure 3: Skeleton of starting differential path for all blocks during the near-collision phase of our CP collision attack on SHA-1 (only the first 20 steps are depicted). The MSB's are on the right and "-" stands for no constraint, while the notation "|" on two bits vertically adjacent mean that these two bits must be equal. The other notations are similar to the ones used in [7]. This is only to give a general idea of the differential path used, as several conditions on the message and/or on the internal state are not represented here.

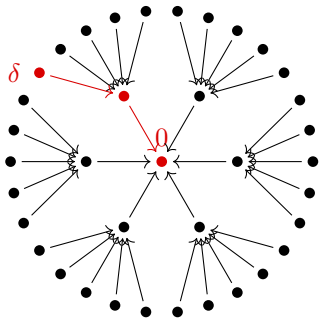

Figure 4: Graph search.

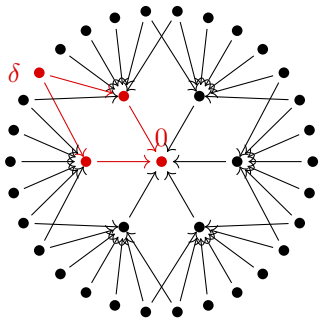

Figure 5: With clustering.

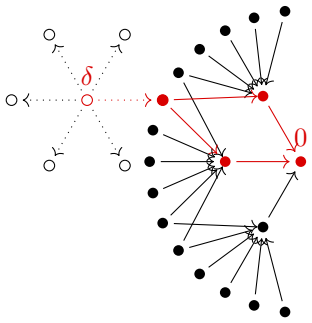

Figure 6: Bi-directional.

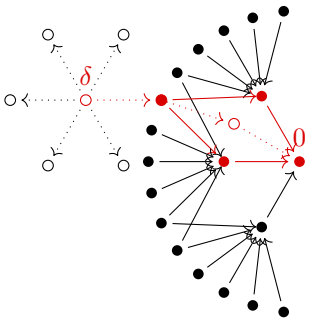

Figure 7: Implicit.

3TB of RAM, we find that almost 90% of the nodes are actually at distance 6 C_{block} or less, as seen in Table 5.

All the differences in this set are active only on a 64-bit mask. Therefore, we use those bit positions for the birthday phase: we truncate SHA-1 to the remaining 96 bits[2] and we generate a large number of partial collisions until one of them corresponds to a difference in the graph.

4.2 Bi-directional Graph

Since the CP collision attack is essentially a path search in a graph, we can use a bi-directional search to make the search more efficient. More precisely, when we evaluate the cost of a node, instead of just looking it up in the graph, we recompute all edges starting from the node to see if they reach the graph and compute the cost using the clustering formula. This corresponds to a bi-directional search where we pre-compute in the backwards direction the set of values that go to zero after at most 10 blocks, and during the online phase, we compute one block forward. This is illustrated by Figure 6, where black dots correspond to precomputed nodes stored in the graph, and white dots are only computed during the online phase.

This can be seen as a time-memory trade-off: we use nodes at a distance up to 11 blocks, but we only build explicitly the graph with 10 blocks. Moreover, we can use nodes that are not reachable with a single trail of cost below 24 C_{block}, and that are therefore excluded from our initial graph. Indeed, if there exists a trail such that the cost is below 24 C_{block} when removing an edge, the forward search using that edge will hit the explicit graph, and we can evaluate the distance of the node.

We can't compute exactly the size of this implicit graph, but we can evaluate it experimentally by simulating the birthday phase of the attack. We found that we need on average $2^{26.4}$ attempts before hitting the graph, which corresponds to a graph size of roughly 2^{38} (assuming that we detect being in the graph with a probability of 0.75, as was the case with the parameters of [12]).

4.3 Implicit Nodes

Following [12], we build the graph using a set $\mathcal{D}$ of 8768 potential output differences with high probability (corresponding to a cost up to 8 C_{block}). However, there are many other output differences that can be useful in our attack, even if they have a lower probability: we can use a block as long as the new state difference gets closer to a collision. Therefore, during the near-collision phase, instead of keeping only blocks with an output difference corresponding to an explicit edge of the graph, we keep all blocks that follow the trail up to step 61 and we look up the new state difference in the graph (using the bi-directional strategy above). With a larger number of usable output differences, the cost of each block decreases (Figure 7).

Again, we can't compute explicitly the complexity of this attack strategy, but we can run simulations. According to our experiments with the graph described above, the average cost of the near-collision phase is only 2 C_{block}, even though most of the nodes in the graph correspond to a cost of 6 C_{block} when following edges that have been explicitly considered.

Finally, we can use this strategy to reduce the number of near-collision blocks used in the attack. In practice, we observed that most of the nodes in our graph can actually be reached with fewer than 11 blocks. In particular, when using output differences that do not correspond to edges of the graph, we often reach an output difference that can be erased with fewer blocks than expected, in particular for the first near-collision blocks.

5 Chosen-Prefix Collision Computation

Even though we managed to reduce the cost of the chosen-prefix collision for SHA-1 to only $2^{63.7}$ SHA-1 evaluations, performing such a large-scale computation remains very expensive. We show that it can be computed with an academic budget, for a total cost much lower than US$ 100k.

5.1 Attack Parameters

Using the idea described in the previous section, we have the following parameters for the attack:

- We use a limit of at most 11 blocks, but we aim for 10 blocks at most for the attack (to fit in a 6144-bit key, see next section);
- The graph $\mathcal{G}$ has size roughly 2^{38}, but it is not computed explicitly;
- The birthday stage is a parallel collision search algorithm (using the distinguished points technique of [28]) with a mask of 96 bits, and we need about $2^{26.4}$ partial collisions on those 96 bits. Therefore the expected complexity of the birthday phase is $\sqrt{\pi 2^{96} 2^{26.4}} \approx 2^{62}$;
- We use chains (consecutive iterations of the function from a starting point during the distinguished points technique) of length 2^{28}, resulting in a data complexity of 1/2 TB to store 2^{34} chains;
- We expect a cost of 2 C_{block} for the near-collision phase.

[2] Given by mask 0x7f000000, 0xfff80001, 0x7ffff000, 0x7ffffc0, 0x7fffffff

Max Cost	1 bl.	2 bl.	3 bl.	4 bl.	5 bl.	6 bl.	7 bl.	8 bl.	9 bl.	10 bl.
1 C_{block}	8.17	8.17	8.17	8.17	8.17	8.17	8.17	8.17	8.17	8.17
2 C_{block}	9.17	16.30	19.92	22.05	23.13	23.95	24.44	24.55	24.62	24.65
3 C_{block}	10.17	17.10	21.76	24.66	26.58	27.95	28.96	29.71	30.31	30.76
4 C_{block}	12.53	18.60	22.97	26.34	28.68	30.35	31.56	32.54	33.29	33.88
5 C_{block}	12.53	19.65	24.18	27.44	29.83	31.65	33.04	34.14	34.90	35.42
6 C_{block}	12.53	19.79	24.81	28.26	30.74	32.62	34.05	35.08	35.67	36.03
7 C_{block}	13.09	20.37	25.30	28.82	31.35	33.24	34.59	35.43	35.86	36.15
8 C_{block}	13.09	20.62	25.72	29.27	31.81	33.65	34.81	35.54	35.92	36.19

Table 5: Size of the set $\mathcal{S}$ with various limits on the maximum cost and on the number of near-collision blocks ($\log_2$).

Complexity estimate. Overall, for the attack parameters chosen, the birthday part costs about $2^{62.05}$ `SHA-1` computations, while the near-collision part is expected to require 1 C_{block} for the last block, and 1 C_{block} in total for the previous blocks.

As explained in **??**, we use the boomerang on $M_6[8]$ for the last block, so that the expected time to find a conforming block can be estimated directly from the figures of Table 4 as $C_{\text{block}} = 2^{48.5}/r$. For the intermediate blocks, we don't use this boomerang, so the rate is reduced to $r/1.9$ but we only require $2^{48.08}$ A_{33}-solutions for one C_{block}. Our simulations show that the total cost for all intermediate blocks is roughly one C_{block}, therefore it will take time $C_{\text{block}} = 1.9 \cdot 2^{48.08}/r$. Finally, we can estimate the total attack time as

$$2^{62.05} \cdot h + \frac{2^{48.5} + 1.9 \cdot 2^{48.08}}{r},$$

with r the A_{33}-solution rate (from Table 4), and h the hash-rate for the birthday phase (from Section 5.3). We give concrete complexity estimates on several GPUs in Table 2. Our chosen-prefix collision attack is roughly four time as expensive as an identical-prefix attack.

5.2 A GPU Cluster

We originally estimated that our attack would cost around US$ 160k by renting GPUs from a cloud provider such as Amazon or Google (using spot or preemptible prices). However, since our computations do not need much communication between the GPUs, nor fancy inter-GPU task scheduling, we can consider renting cheaper GPUs from providers that use gaming or mining cards in consumer-grade PCs, rather that the datacenter-grade hardware used by big cloud providers. Services like `gpuserversrental.com` rent GTX 1060 or GTX 1080 GPUs for a price below 5 cents per month per CUDA core; which would give a total cost around US$ 75k to compute a chosen-prefix collision.

Our cluster was made of 150 machines with 6 GPU each (with a mix of GTX 1060 3G, and GTX 1060 6G), and one master node with two 2TB hard drives in a RAID configuration. The master node had a Core i7 CPU, but the GPU nodes had low-end Pentium or Celeron CPU with two cores. Each machine ran Ubuntu Linux, but there was no cluster management software installed (we used `clush` to run commands on all the nodes). We negotiated a price of US$ 37.8k per month (higher than current prices), and used the cluster for two months.

Cost analysis. We paid US$ 75.6k for our computation, but the cost could be as low as US$ 50k with currently lower GPU prices and less idle time. With the same methods, computing an identical-prefix `SHA-1` collision would cost only about US$ 11k. This is clearly within reach of reasonable attackers.

Of course the underlying weakness of `SHA-1` has always been present, even if it was not public (and maybe not discovered). We estimate that a PS3 cluster (as used by Stevens et al. [27], and as deployed by the US army[3]) could have implemented this attack for a cost of a few million dollars in 2010, when SHA-1 was still the most widely used hash function. This underlines that the deprection process of SHA-1 should have been much faster after the publication of the first theoretical collision attack in 2004.

Looking at the future, this attack will get even cheaper as computation costs decrease. Following Moore's law (that seems to be still valid for GPU[4]), we estimate that it should cost less than US$ 10k to generate a `SHA-1` chosen-prefix collision by 2025.

5.3 Birthday Phase

In order to simplify the implementation, we implemented the birthday phase with two distinct steps: in the first step, each GPU computes independently a series of

[3] https://phys.org/news/2010-12-air-playstation-3s-supercomputer.html

[4] https://blogs.nvidia.com/blog/2017/05/10/nvidia-accelerates-ai-launches-volta-dgx-workstation-robot-simulator-more/

Date	Event	Complexity	# collisions
July 25	Starting cluster setup		
July 27	Computation started		
August 14	Step 2 unsuccessful	$2^{61.9}$	$2^{25.8}$
August 20	Step 2 unsuccessful	$2^{62.4}$	$2^{26.6}$
August 24	Step 2 unsuccessful	$2^{62.6}$	$2^{27.1}$
August 30	Step 2 successful!	$2^{62.9}$	$2^{27.7}$

Table 6: Timeline of the birthday phase.

Date	Event	$\#A_{61}$-sol	Complexity
September 07	Block 1 found[a]	2^{16}	0.11 C_{block}
September 09	Block 2 found	$2^{13.5}$	0.02 C_{block}
September 13	Block 3 found	$2^{16.9}$	0.21 C_{block}
September 14	Block 4 found	$2^{10.8}$	0.003 C_{block}
September 16	Block 5 found	$2^{15.5}$	0.08 C_{block}
September 18	Block 6 found	$2^{15.5}$	0.08 C_{block}
September 20	Block 7 found	2^{16}	0.11 C_{block}
September 21	Block 8 found	$2^{14.5}$	0.04 C_{block}
September 27	Block 9 found[b]	$2^{18.2}$	0.38 C_{block}

Table 7: Timeline of the near-collision phase. C_{block} corresponds to $2^{19.17}$ A_{61}-solutions, excepted for the last block where the use of an extra boomerang increases it to $2^{19.58}$

[a] Two solutions found
[b] Using the $M_6[8]$ boomerang

chains, and in the second step we gather all the results, sort them to find collisions in the end-points, and re-run the chain to locate the collisions. Our implementation runs at a speed of $h = 3.5$GH/s on GTX 1060 GPUs (respectively 3.2 GH/s on GTX 970 and 11 GH/s on GTX 1080 Ti). This is somewhat lower than the hashcat benchmarks reported in Table 2 because hashcat can skip some parts of `SHA-1`, and we have to keep two `SHA-1` states in the registers to implement the birthday phase. Every time we run the second step, we then search the collisions in the graph, to determine whether we have reached a useful starting point (this is run on a separate machine with at least 1TB or RAM, and we let the cluster restart the first step in the meantime).

As shown in Table 6, we ran step 2 four times, and we have been quite unlucky in the birthday phase, only succeeded after finding $2^{27.7}$ collisions, rather than the estimated $2^{26.4}$. It took us 34 days to compute those chains, which corresponds to a hashrate 2.9 TH/s for our cluster (including downtime).

5.4 Near-collision Phase

The near-collision phase is very technical and very complex. Every time a block is found, we have to prepare the search for the next block. This first requires to traverse the graph $\mathcal{G}$ to find the parameters for the next block: we have different constraints in the last steps depending on which output differences are desired. Then, we had to generate a new non-linear part for the early steps, as explained in Section 3.4. We used tools similar to [7], which take a lot of parametrization and trial-and-error to have a proper non-linear part that fits nicely with the core differential path.

This was automated to some extent, but still took between a few hours and a few days of manual work to prepare for each block (it took more time for the first blocks because there are more constraints to build the path, and we were more experienced for the later blocks). Unfortunately, this means that the GPU cluster was not doing useful work during this time. We remark that our attack could have cost less if we had fully automated the entire cryptanalysis process, or if we had improved the search algorithm for the non-linear part of the differential path. This is definitely not impossible to achieve, but it would require a lot of tedious work.

For the last block, we started the computation without the boomerang on $M_6[8]$, and modified the path and the code after one day to include it. As explained in Section 3, this extra boomerang reduces the quality of A_{61}-solutions, so that we need $4/3$ times the number of solutions ($2^{19.58}$ instead of $2^{19.17}$), but it almost doubles the production rate of these solutions. In total, this reduces the computation time by a factor $1.9/4/3 \approx 1.4$.

As expected, intermediate blocks cost much less than C_{block} (the cost of a block with a pre-determined output difference) because we can target a large number of output differences. Only the last block is expected to cost C_{block}. However, we were quite lucky in this phase of attack, because we found all the blocks after only 0.9 C_{block}, rather than the estimated 2 C_{block}. In particular, the last block was found after only $2^{18.2}$ A_{61}-solutions (0.38 C_{block}), instead of the expected $2^{19.58}$.

A timeline of the near-collision phase is given in Table 7, and the chosen-prefix collision is given in the full version of the paper [13].

5.5 Resources Used

A quick overview of the resources used for each part is given in Table 8. If we evaluate the total useful GPU time spent for the attack, we have roughly 78 years for the birthday phase, 25 years for blocks 1 to 9, and 10 years for the last block. This means that roughly 75% of our GPU time was useful. If we convert the attack time

to SHA-1 evaluations, we arrive at a total of $2^{63.6}$, which is quite close to the estimate of $2^{63.5}$ given in Table 2.

6 Application to PGP Web of Trust

Our demonstration of a chosen-prefix collision targets the PGP/GnuPG Web of Trust. More precisely, our goal is to create two PGP keys with different UserIDs, so that key B is a legitimate key for Bob (to be signed by the Web of Trust), but the signature can be transferred to key A which is a forged key with Alice's ID. This will succeed if the hash values of the identity certificates collide, as in previous attacks against X.509 MD5-based certificates [25, 27]. Moreover, due to details of the PGP/GnuPG certificate structure, our attack can reuse a single collision to target arbitrary users Alice and Bob: for each victim, the attacker only needs to create a new key embedding the collision, and to collect a SHA-1 signature. This is arguably the first practical attack against a real world security application using weaknesses of SHA-1.

6.1 Exploiting a Chosen-prefix Collision

We now focus on the identity certificates that will be hashed and signed. Following RFC 4880 [5], the hash computation done during certificate signing receives the public key packet, then a UserID or user attribute packet, and finally a signature packet and a trailer. The idea of the attack is to build two public keys of different sizes, so that the remaining fields to be signed are misaligned, and we can hide the UserID of key A in another field of key B. Following RFC 4880, the signature packet is protected by a length value at the beginning *and at the end*, so that we have to use the same signature packet in key A and key B (we cannot stuff data in the hashed subpacket). Therefore, we can only play with the UserID and/or user attribute packets. Still, a user attribute packet with a JPEG image gives us enough freedom to build colliding certificates, because typical JPEG readers ignore any bytes after the End of Image marker (`ff d9`). This gives us some freedom to stuff arbitrary data in the certificate.

More precisely, we build keys A and B as follows. Key A contains an 8192-bit RSA public key, and a UserID field corresponding to Alice. On the other hand, key B contains a 6144-bit RSA public key, the UserID of Bob and a JPEG image. Therefore, when Bob gets a certification signature of his key, the signer will sign two certificates: one containing his public key and UserID, and another one containing the public key and the image. The public keys A and B and the image are crafted in such a way to generate a collision between the certificates with the key A and Alice's UserID, and the certificate with key B and the image.

6.1.1 Content of Identity Certificates

Figure 8 shows a template of the values included in the identity certificate: those values are hashed when signing a key, and we want the two hashes to collide. In this example, the UserID field of key A contains "`Alice <alice@example.com>`", and the image in key B is a valid JPEG image that will be padded with junk data after the End of Image marker. The real JPEG file is 181 bytes long[5] (from `ff d8` to `ff d9`), and it is padded with 81 bytes, so that the file included in the key is 262 bytes long (here the padding includes 46 bytes corresponding to the end of the modulus of key A, 5 bytes corresponding to the exponent of key A, and 30 bytes corresponding to Alice's UserID).

In Figure 8, we use the following symbols:

- `01` Bytes with a fixed value are fixed by the specifications, or chosen in advance by the attacker (length of fields, UserID, user attribute, ...)
- `??` Represent bytes that are determined by the chosen-prefix collision algorithm (the messages M and M' to generate a collision)
- `!!` Represent bytes that are selected after finding the collision, to generate an RSA modulus with known prime factors
- `..` Represent bytes that are copied from the other certificate
- `**` Represent time-stamps chosen by the attacker
- `$$` Represent the time-stamp chosen by the signer

Underlined values correspond to packet headers (type and length).

6.1.2 Attack Procedure

To carry out the attack, we have to perform the following steps:

1. Build a chosen-prefix collision with prefixes "`99 04 0d 04 ** ** ** ** 01 20 00`" and "`99 03 0d 04 ** ** ** ** 01 18 00`", after filling the `**` with two arbitrary time-stamps. The chosen-prefix collision must have at most 10 near-collision blocks. This determines the `??` bytes of the keys.
2. Choose a tiny JPEG image to include in key B (fixed orange bytes), and an arbitrary UserID to include in key A (fixed yellow bytes)
3. Select "`!!`" bytes in B to obtain a modulus with known factors
4. Select "`!!`" bytes in A to obtain a modulus with known factors
5. Generate key B with the modulus and the padded JPEG. Ask for a signature of the key.

[5] Building a JPEG image smaller than 256 bytes is not easy, but it is possible

Phase	Step	Main resource	Repetitions	Wall time
Setup	Preparation of the graph	CPU and RAM		$\approx$ 1 month
Birthday	Computing chains	GPU		34 days
	Sorting chains	Hard drive	4 ×	$\approx$ 1 day
	Locating collisions	GPU	4 ×	$<$ 1/2 day
	Searching in graph	RAM	4 ×	$<$ 1/2 day
Blocks	Building trail & code	Human Time	9 ×	$\approx$ 1 day
	Finding block	GPU	8 ×	3 hours – 3 days
	Checking results in graph	RAM	8 ×	$<$ 1/2 hour
	Finding last block	GPU	1 ×	6 days

Table 8: Resources used for the attack

6. Copy the signature to key A.

We point out that the chosen-prefix collision is computed *before* choosing the UserIDs and images that will be used in the attack. Therefore, a single CPC can be reused to attack many different victims. This contrasts with attacks on X.509 certificates [25, 27], where the identifier is hashed before the public key.

In order to build the modulus (steps 3 and 4 above), we use the same strategy as in previous works [25, 27]. More precisely, the high order bits are fixed by previous steps, and the low-order bits can be chosen freely. Therefore we have to find a modulus in an interval $[A, B]$ with a known factorisation. We select a random prime P (in the order of $B - A$), and we compute $Q = \lfloor B/P \rfloor$. If Q is a prime, we use $P * Q$ as the modulus: we have $A \leq P * Q \leq B$ when $P \leq B - A + 1$. This takes a few minutes in practice.

We note that the factors of the modulus are unbalanced. With the template of Figure 8, we expect factors of 88 bits and 6056 bits for Key B, and 368 bits and 7824 bits for key A. In practice we managed to find a CP collision with fewer blocks than in Figure 8, so that key B actually has factors of 1112 bits and 5032 bits. This makes both keys hard to factor. As mentioned in [14], it is possible to find modulus with somewhat larger factors using more advanced techniques.

6.1.3 Example Keys

An example of a pair of keys generated with this procedure can be directly downloaded from these URLs:

Key A: `https://SHA-mbles.github.io/alice.asc`

Key B: `https://SHA-mbles.github.io/bob.asc`

The keys can be examined with `pgpdump -i` to see that they include the same signature.

In our demonstration, we chose a time-stamp far in the future to avoid malicious usage of our collision. However, an attacker that can repeat our work will obviously use a valid time-stamp.

6.1.4 Attack Variant

We also found an alternative attack, exploiting the PGP key format in a slightly different way, where key B contains a short public key followed by a JPEG image. We would consider both the public key and the image as the prefix, and stuff the CPC blocks inside the image (after the EOI marker). This variant leaves a smaller space for the CPC blocks, but the advantage is that key A is less suspicious because it doesn't need to contain a valid JPEG file inside the modulus (the modulus is really made of random-looking blocks). On the other hand, this variant requires to compute a new CPC for each key B.

6.2 Impact

As explained in Section 7.1, the "classic" branch of GnuPG (v1.4) uses `SHA-1` by default for identity certifications, and there is still a non-negligible number of keys signed with `SHA-1`. Before our attack was disclosed, `SHA-1` signatures were also accepted by the "modern" branch of GnuPG (v2.2). This made the attack usable in practice.

In addition, a single CPC can be reused to attack many different victims, so that the cost of the CPC is just a one-off cost. Given our cost estimation around US$ 50k, this is well within reach of strong adversaries.

7 `SHA-1` Usage and Disclosure

`SHA-1` is still used in a surprising number of security applications. It is supported in many secure channel protocols (TLS, SSH), and remains actually used for some fraction of the connections. It is also used for PGP identity certifications, and it is the foundation of GIT versioning system. We expect there are also an important

	Key A (RSA-8192)		Key B (RSA-6144)
0x0000	99 04 0d 04 ** ** ** ** 01 20 00 ?? ?? ?? ?? ??		99 03 0d 04 ** ** ** ** 01 18 00 ?? ?? ?? ?? ??
	?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ??		?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ??
	?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ??		?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ??
	?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ??		?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ??
0x0040	?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ??		?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ??
	?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ??		?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ??
	?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ??		?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ??
	?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ??		?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ??
	⋮		⋮
0x02c0	?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ??		?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ??
	?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ??		?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ??
	?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ??		?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ??
	?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ??		?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ??
	Collision here!		
0x0300	.. ..	←	!! !! !! !! !! !! !! !! !! !! !! 00 11 01 00 01
	.. ..	←	d1 00 00 01 19 c0 57 01 10 00 01 01 00 00 00 00
	.. ..	←	00 00 00 00 00 00 00 00 ff d8 ff db 00 43 00 ff
	.. ..	←	ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
0x0340	.. ..	←	ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
	.. ..	←	ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
	.. ..	←	ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
	.. ..	←	c0 00 0b 08 00 40 00 58 01 01 11 00 ff c4 00 28
0x0380	.. ..	←	00 01 01 01 00 00 00 00 00 00 00 00 00 00 00 00
	.. ..	←	00 00 04 03 10 01 00 00 00 00 00 00 00 00 00 00
	.. ..	←	00 00 00 00 00 00 ff da 00 08 01 01 00 00 3f 00
	.. ..	←	d0 4e a0 01 3a 80 04 ea 01 3a 80 04 e0 00 a0 13
0x03c0	.. ..	←	8a 13 82 84 e2 84 e0 00 00 28 4e 00 0a 13 8a 13
	.. .. !! !! !!	↔	a8 00 4e a1 3a 80 4e 28 4e 28 07 ff d9 .. .. ..
	!! !! !! !! !! !! !! !! !! !! !! !! !! !! !! !!	→	.. ..
	!! !! !! !! !! !! !! !! !! !! !! !! !! !! !! !!	→	.. ..
0x0400	!! !! !! !! !! !! !! !! !! !! !! 00 11 01 00 01	→	.. ..
	b4 00 00 00 19 41 6c 69 63 65 20 3c 61 6c 69 63	→	.. ..
	65 40 65 78 61 6d 70 6c 65 2e 63 6f 6d 3e 04 10	→	.. .. 04 10
	01 02 00 06 05 02 .. .. 04 ff 00 00 00 0c	←	01 02 00 06 05 02 $$ $$ $$ $$ 04 ff 00 00 00 0c

Figure 8: Construction of colliding OpenPGP identity certificates. The colour corresponds to the packets hashed when computing the signature: first, the public key packet (with header), then the UserID or user attribute, and finally the signature packet and trailer. Arrows show when a value is chosen in one key and copied to the other.

number of proprietary systems using SHA-1, but getting actual data on this is difficult.

Collisions and chosen-prefix collisions do not threaten all those usages (in particular HMAC-SHA-1 seems relatively safe), but there are several settings that are directly affected by chosen-prefix collisions:

- PGP identities can be impersonated if trusted third parties sign identity certificates with SHA-1 (see 7.1)
- X.509 certificates could be broken if some CAs issue SHA-1 certificates with predictable serial numbers (see 7.2)
- TLS and SSH connections using SHA-1 signatures to authenticate the handshake could be attacked with the SLOTH attack [1] if the CP collision can be generated extremely quickly (see 7.3 and 7.4)

We stress that when a protocol supports several hash functions, those attacks are possible as long as SHA-1 is *supported* by implementations, even if it is not selected during normal use. A man-in-the-middle attacker will just force the parties to use SHA-1.

More generally, as cryptographers, we recommend to deprecate SHA-1 everywhere, even when there is no direct evidence that this weaknesses can be exploited. SHA-1 has been broken regarding collision resistance for 15 years, and there are better alternatives available, well-studied, and standardized (SHA-2 [17], SHA-3 [18]). There is no good reason to use SHA-1 in modern security software. Attacks only get better over time, and the goal of the cryptanalysis effort is to warn users so that they can deprecate algorithms *before* the attacks get practical.

As a stopgap measure, the collision-detection library of Stevens and Shumow [26] can be used to detect attack attempts (it successfully detects our attack).

Responsible disclosure. We have tried to contact the authors of affected software before announcing this attack, but due to limited resources, we could not notify everyone. We detail below the main affected products, some of the responses we received, and countermeasures deployed at the time of writing. More up to date information will be available on the website of the attack: https://sha-mbles.github.io.

7.1 SHA-1 Usage in GnuPG

There are currently two supported branches of GnuPG: GnuPGv1 is the "legacy" (or "classic") branch, and GnuPGv2 is the "modern" branch. The first version of GnuPGv2 dates back to 2006, and the "legacy" branch is no longer recommended, but the transition took a long time. In particular, GnuPGv1 was still the default version in Fedora 29 (released in October 2018), and in Ubuntu 16.04 LTS (which is supported until April 2021).

GnuPG supports many different algorithms, including SHA-1. Moreover, SHA-1 is the default algorithm for identity certification in GnuPGv1. This is why we targeted PGP in our demonstration of chosen-prefix collisions. After we disclosed our results to the GnuPG team, SHA-1 signatures have been deprecated in the GnuPGv2 branch (commit edc36f5, CVE-2019-14855).

Web of Trust. The original trust model of PGP was the Web of Trust. Instead of using a central PKI, users sign each other's keys to attest of their identity (*e.g.* when attending a key signing party), and trust such certificates from third parties. A scan of the PGP Web of Trust (i.e. identity certifications on public keyservers) shows that roughly 1% of the identity certifications issued in 2019 use SHA-1. This probably corresponds to usage of GnuPGv1 with the default settings, and would make our attack feasible.

CAcert. CAcert (http://cacert.org/) is one of the main CAs for PGP keys, and they still use SHA-1 to sign user keys. We have first contacted them by email on December 14th, and got an answer on January 6th acknowledging this issue. They are now planning a switch to a secure hash function for key certification.

7.2 SHA-1 Usage in X.509 Certificates

The CA/Browser Forum decided to sunset SHA-1 in October 2014, and its members are not supposed to issue SHA-1 certificates after 2016. Web browsers have enforced similar rules, and all modern browsers now reject SHA-1 certificates.

However, SHA-1 certificates are still present for legacy purposes, on services that are used by older clients that can not be upgraded. In particular, it remains possible to buy a SHA-1 certificate today, and there are a few recently-issued certificates in use on the web[6]. There are also a few old SHA-1 certificates still in use[7]. Those certificates are rejected by modern web browsers, but they can be accepted by non-web TLS clients. For instance, it seems that the Mail application in Windows 10 can open an IMAP session secured with a SHA-1 certificate without warning.

Chosen-prefix collisions against MD5 have been able to break the security of certificates in the past, with the creation of a Rogue CA by Stevens *et al.* [27], and in the wild by the flame malware [21]. If some of the CAs still issuing SHA-1 certificates use predictable serial numbers, a similar attack might be possible today (being located at the beginning of the "to-be-signed" part of the certificate, if the serial number is unpredictable then the CP collision attack is thwarted as a crucial part of the hashed input is not controlled by the attacker).

7.3 SHA-1 Usage in TLS

Besides certificates, there are two places where SHA-1 can be used in the TLS protocol: SHA-1 can be used to sign the handshake, and HMAC-SHA-1 can be used to authenticate data in the record protocol.

Handshake. Client authentication in TLS uses a signature of the transcript, which can be abused using CP collisions, as shown by the SLOTH attacks [1]. However, this remains far from being a practical attack, because

[6] Some examples can be found by searching through certificate transparency logs: http://web.archive.org/web/20191227165750/https://censys.io/certificates?q=tags%3Atrusted+AND+parsed.signature.signature_algorithm.name%3ASHA1%2A+AND+parsed.validity.start%3A%5B2019-01-01+TO+%2A%5D

[7] As seen in this scan: http://web.archive.org/web/20191227165038/https://censys.io/ipv4?q=443.https.tls.validation.browser_trusted%3AYes+AND+443.https.tls.certificate.parsed.signature_algorithm.name%3ASHA1%2A

the CP collision has to be computed in a very short time (timeout value is generally set to a few seconds, but can be up to several minutes).

In TLS 1.0 and 1.1, the handshake is hashed with the concatenation of `SHA-1` and `MD5`. Using the multicollision attack from Joux [9], computing a CP collision for `MD5` $\|$ `SHA-1` is not much harder than for `SHA-1`. We give concrete figures in Table 2, showing that this is probably within reach of a well motivated adversary.

In TLS 1.2, the hash function used is configurable. The vast majority of TLS 1.0/1.1 clients and server support `SHA-1`, and many servers actually *prefer* to use `SHA-1`, even when the client offers better algorithms[8,9].

In TLS version 1.3, `MD5` and `SHA-1` have been removed.

Ciphersuites. The large majority of clients and servers support ciphersuites where `HMAC-SHA-1` is used to authenticate the packets, at least for interoperability reasons. It seems that usage of `HMAC-SHA-1` represents a few percent of all the connections[10,11]. This usage is not threatened by our attack, but we recommend to avoid `SHA-1` usage when possible.

OpenSSL. The next version of OpenSSL will no longer allow X.509 certificates signed using `SHA-1` at security level 1 and above (commit `68436f0`). Since security level 1 is the default configuration for TLS/SSL, this will prevent `SHA-1` usage for certificates.

Debian Linux had previously set the default configuration to security level 2 (defined as 112-bit security) in the latest release (Debian Buster); this already prevents dangerous usage of `SHA-1` (for certificates and handshake signature).

7.4 `SHA-1` Usage in SSH

`SHA-1`'s usage in SSH is similar to its usage in TLS. The SSH-2 protocol supports usage of `SHA-1` to sign the transcript (at the end of the key exchange), and `HMAC-SHA-1` to authenticate the data in the record protocol. As in the TLS case, usage of `SHA-1` to sign the transcript has been shown to be potentially vulnerable to the SLOTH attack [1], but this is not practical given the timing constraints (usually just a few seconds, but can be configured to a longer period of time).

Again, the choice of cryptographic algorithms depends on a negotiation between the client and server, so it is hard to know exactly what will be selected. However, scans of the IPv4 space from censys at the time of writing show that roughly 17% of servers use `SHA-1` to sign the transcript[12], and 9% of servers use `HMAC-SHA-1` in the record protocol[13]. This mostly corresponds to servers running old versions of SSH daemons.

OpenSSH. Due to our results, since version 8.2 of OpenSSH a "future deprecation notice" is included, explaining that `SHA-1` signatures will be disabled in the near-future.

7.5 Other Usages of `SHA-1`

DNSSEC. `SHA-1` is still used in DNSSEC, with 18% of the top-level domains using `SHA-1` at the time of writing[14]. Since DNSSEC signatures include user-supplied content, CP collisions could be used to attack the DNSSEC system.

GIT. GIT relies heavily on `SHA-1` to identify all objects in a repository. It does not necessarily require cryptographic security from `SHA-1`, but there are certainly some attack scenarios where attacks on `SHA-1` would matter. In particular, signed GIT commits are essentially signatures of a `SHA-1` hash, so they would be sensitive to collision attacks.

The GIT developers have been working on replacing SHA-1 for a while[15], and they use a collision detection library [26] to mitigate the risks of collision attacks.

Timestamping. Many timestamping servers apparently support `SHA-1`, such as: https://sectigo.com/resources/time-stamping-server

8 Conclusion and Future Works

This work shows once and for all that `SHA-1` should not be used in any security protocol where some kind of collision resistance is to be expected from the hash function. Continued usage of `SHA-1` for certificates or for

[8] http://web.archive.org/web/20191227174651/https://censys.io/domain/report?field=443.https.tls.signature.hash_algorithm

[9] http://web.archive.org/web/20191227174551/https://censys.io/domain?q=443.https.tls.signature.hash_algorithm%3Asha1

[10] See https://telemetry.mozilla.org/new-pipeline/dist.html#!measure=SSL_CIPHER_SUITE_FULL, were buckets 5, 61 and 63 correspond to `HMAC-SHA-1` ciphersuites

[11] http://web.archive.org/web/20191226134753/https://censys.io/domain/report?field=443.https.tls.cipher_suite.name.raw

[12] http://web.archive.org/web/20191226130952/https://censys.io/ipv4/report?field=22.ssh.v2.selected.kex_algorithm

[13] http://web.archive.org/web/20191226131928/https://censys.io/ipv4/report?field=22.ssh.v2.selected.client_to_server.mac

[14] https://www.dns.cam.ac.uk/news/2020-02-14-sha-mbles.html

[15] https://git-scm.com/docs/hash-function-transition/

authentication of handshake messages in TLS or SSH is dangerous, and there is a concrete risk of abuse by a well-motivated adversary. `SHA-1` has been broken regarding collision resistance since 2004, but it is still used in many security systems. We strongly advise users to remove `SHA-1` support to avoid downgrade attacks.

We also show that gaming or mining GPUs offer a cheap and efficient way to attack symmetric cryptography primitives. In particular, it now costs less than US$ 100k to rent GPUs and break cryptography with a security level of 64 bits (i.e. to compute 2^{64} operations of symmetric cryptography).

The cost of our attack is roughly four times the cost of a plain collision attack, so there is limited room for improvements in terms of complexity.

On the other hand, we believe there is some possibility to reduce the number of blocks used in the attack without increasing the complexity much. Firstly, with a better use of the global parameters of the general chosen-prefix collision attack. By playing with the number of blocks, the allowable probabilities and the size of the graph, one could probably find a better configuration. Secondly, by not considering only the core differential trail from [23], but using other interesting ones, we would increase the pool of available differences and in turn reduce the required number of blocks.

Acknowledgements

The authors would like to thank Vesselin Velichkov for his help with regards to an initial analysis of neutral bits applicability on `SHA-1` and Werner Koch for his comments on the applicability of our attacks on PGP. The authors would also like to thank `gpuserversrental.com` for their efficient service regarding the GPU cluster renting. The second author is supported by Temasek Laboratories, Singapore.

A small part of the experiments presented in this paper were carried out using the Grid'5000 testbed, supported by a scientific interest group hosted by Inria and including CNRS, RENATER and several Universities as well as other organizations (see `https://www.grid5000.fr`). Development and small scale experiments before launching the main computation were carried out on the rioc cluster from Inria.

References

[1] Karthikeyan Bhargavan and Gaëtan Leurent. Transcript collision attacks: Breaking authentication in TLS, IKE and SSH. In *NDSS 2016*. The Internet Society, February 2016.

[2] Eli Biham and Rafi Chen. Near-collisions of SHA-0. In Franklin [8], pages 290–305.

[3] Eli Biham, Rafi Chen, Antoine Joux, Patrick Carribault, Christophe Lemuet, and William Jalby. Collisions of SHA-0 and reduced SHA-1. In Ronald Cramer, editor, *EUROCRYPT 2005*, volume 3494 of *LNCS*, pages 36–57. Springer, Heidelberg, May 2005.

[4] Gilles Brassard, editor. *CRYPTO*, volume 435 of *Lecture Notes in Computer Science*. Springer, 1990.

[5] J. Callas, L. Donnerhacke, H. Finney, D. Shaw, and R. Thayer. *RFC 4880 - OpenPGP Message Format*. Internet Activities Board, November 2007.

[6] Ivan Damgård. A Design Principle for Hash Functions. In Brassard [4], pages 416–427.

[7] Christophe De Cannière and Christian Rechberger. Finding SHA-1 characteristics: General results and applications. In Xuejia Lai and Kefei Chen, editors, *ASIACRYPT 2006*, volume 4284 of *LNCS*, pages 1–20. Springer, Heidelberg, December 2006.

[8] Matthew Franklin, editor. *CRYPTO 2004*, volume 3152 of *LNCS*. Springer, Heidelberg, August 2004.

[9] Antoine Joux. Multicollisions in iterated hash functions. Application to cascaded constructions. In Franklin [8], pages 306–316.

[10] Antoine Joux and Thomas Peyrin. Hash functions and the (amplified) boomerang attack. In Alfred Menezes, editor, *CRYPTO 2007*, volume 4622 of *LNCS*, pages 244–263. Springer, Heidelberg, August 2007.

[11] Vlastimil Klima. Tunnels in hash functions: MD5 collisions within a minute. Cryptology ePrint Archive, Report 2006/105, 2006. `http://eprint.iacr.org/2006/105`.

[12] Gaëtan Leurent and Thomas Peyrin. From collisions to chosen-prefix collisions application to full SHA-1. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part III*, volume 11478 of *LNCS*, pages 527–555. Springer, Heidelberg, May 2019.

[13] Gaëtan Leurent and Thomas Peyrin. SHA-1 is a Shambles - First Chosen-Prefix Collision on SHA-1 and Application to the PGP Web of Trust. Cryptology ePrint Archive, Report 2020/014, 2020. `https://eprint.iacr.org/2020/014`.

[14] Marc Stevens. Attacks on Hash Functions and Applications. PHD Thesis, Leiden University, June 2012.

[15] Ralph C. Merkle. One Way Hash Functions and DES. In Brassard [4], pages 428–446.

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

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

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

[19] Ronald L. Rivest. The MD4 message digest algorithm. In Alfred J. Menezes and Scott A. Vanstone, editors, *CRYPTO'90*, volume 537 of *LNCS*, pages 303–311. Springer, Heidelberg, August 1991.

[20] Ronald L. Rivest. *RFC 1321: The MD5 Message-Digest Algorithm*. Internet Activities Board, April 1992.

[21] Marc Stevens. Counter-cryptanalysis. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part I*, volume 8042 of *LNCS*, pages 129–146. Springer, Heidelberg, August 2013.

[22] Marc Stevens. New collision attacks on SHA-1 based on optimal joint local-collision analysis. In Thomas Johansson and Phong Q. Nguyen, editors, *EUROCRYPT 2013*, volume 7881 of *LNCS*, pages 245–261. Springer, Heidelberg, May 2013.

[23] Marc Stevens, Elie Bursztein, Pierre Karpman, Ange Albertini, and Yarik Markov. The first collision for full SHA-1. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part I*, volume 10401 of *LNCS*, pages 570–596. Springer, Heidelberg, August 2017.

[24] Marc Stevens, Pierre Karpman, and Thomas Peyrin. Freestart collision for full SHA-1. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part I*, volume 9665 of *LNCS*, pages 459–483. Springer, Heidelberg, May 2016.

[25] Marc Stevens, Arjen K. Lenstra, and Benne de Weger. Chosen-prefix collisions for MD5 and colliding X.509 certificates for different identities. In Moni Naor, editor, *EUROCRYPT 2007*, volume 4515 of *LNCS*, pages 1–22. Springer, Heidelberg, May 2007.

[26] Marc Stevens and Daniel Shumow. Speeding up detection of SHA-1 collision attacks using unavoidable attack conditions. In Engin Kirda and Thomas Ristenpart, editors, *USENIX Security 2017*, pages 881–897. USENIX Association, August 2017.

[27] 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. In Shai Halevi, editor, *CRYPTO 2009*, volume 5677 of *LNCS*, pages 55–69. Springer, Heidelberg, August 2009.

[28] Paul C. van Oorschot and Michael J. Wiener. Parallel collision search with cryptanalytic applications. *Journal of Cryptology*, 12(1):1–28, January 1999.

[29] Xiaoyun Wang, Yiqun Lisa Yin, and Hongbo Yu. Finding collisions in the full SHA-1. In Victor Shoup, editor, *CRYPTO 2005*, volume 3621 of *LNCS*, pages 17–36. Springer, Heidelberg, August 2005.