SHA-1 is a Shambles

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

Gaëtan Leurent[1] and Thomas Peyrin[2,3]

[1] Inria, France
[2] Nanyang Technological University, Singapore
[3] Temasek Laboratories, Singapore
gaetan.leurent@inria.fr, thomas.peyrin@ntu.edu.sg

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

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 [WYY05], but due to its high complexity it was only implemented in practice in 2017, using a large GPU cluster [SBK+17]. More recently, an almost practical *chosen-prefix* collision attack against SHA-1 has been proposed [LP19]. 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 collisions attack against SHA-1: on an Nvidia GTX 970, identical-prefix collisions can now be computed with a complexity 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 11k US$ for a collision, and 45k US$ 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 (we paid 75k US$ because GPU prices were higher, and we wasted some time preparing the attack).

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). 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 a forgery. This proves that SHA-1 signatures now offers 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).

Keywords: SHA-1 · Cryptanalysis · Chosen-prefix collision · HPC · GPU · PGP · GnuPG

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, in order to improve efficiency and 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, aka 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 has then usually 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 for example). Thus, one can consider stronger and more relevant variants of the collision attack in practice, such as the so-called *chosen-prefix* collision [SLdW07] 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 $H(P \parallel M) = H(P' \parallel M')$, where $\parallel$ denotes concatenation. With such ability, the attacker can obtain a collision even though prefixes can be chosen arbitrarily (and thus potentially contain some 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 random and completely uncontrolled internal differences created by the prefixes. Yet, a CP collision attack was found for the MD5 hash function [SLdW07], eventually leading to the creation of colliding X.509 certificates, and later of a rogue Certificate Authority (CA) [SSA+09]. CP collisions have also been shown to break important internet protocols, including TLS, IKE, and SSH [BL16], because they allow forgeries of the handshake messages.

Largely inspired by MD4 [Riv91] and then MD5 [Riv92], SHA-1 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 disallowed to be used for digital signatures at the end of 2013). Indeed, even though his successors SHA-2 or SHA-3 are believed to be secure, SHA-1 has been broken by a theoretical collision attack in 2004 [WYY05]. 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 [SBK+17]. Because it took more than a decade to compute an actual collision and because plain collisions are difficult to use directly to attack a protocol, the on-field SHA-1 deprecation process has been quite slow in practice and one can still observe many uses of SHA-1 in the wild unfortunately, migration being expensive. Web browsers have recently started to reject certificates with SHA-1 signatures, but there are still many users with older browsers, and many protocols and software that allow SHA-1 signatures. As observed in [LP19], it is still possible to buy a SHA-1 certificate from a trusted CA, many email clients accept a SHA-1 certificate when opening a TLS connection, and SHA-1 is also widely supported to authenticate TLS handshake messages.

Very recently, a CP collision attack against SHA-1 has been published [LP19], 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 prefix 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 [SLdW07, Ste13b]). 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 deploy as the attack contains many very technical parts.

Table 1: Comparison of previous and new cryptanalysis results on SHA-1. A free-start collision is a collision or 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).

Function	Collision type	Complexity	Ref.
SHA-1	free-start collision	$2^{57.5}$	[SKP16]
	collision	2^{69}	[WYY05]
		$2^{64.7}$	[Ste13b, SBK+17][a]
		$2^{61.2}$	New
	chosen-prefix collision	$2^{77.1}$	[Ste13b]
		$2^{67.1}$	[LP19]
		$2^{63.4}$	New

[a]The attack has a complexity of 2^{61} on CPU, and $2^{64.7}$ on GPU

1.1 Our Contributions

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

Complexity improvements. While the work of [LP19] 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 neutral bits [BCJ+05] and boomerangs [JP07] on state-of-the-art collision attacks for SHA-1 (focusing on the near-collision block search). 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 $2^{64.7}$. We note that the general ideas underlying these improvements might be interesting for other cryptanalysis than SHA-1 one.

Second, we improved the graph-based technique of [LP19] to compute a chosen-prefix collision. Using a larger graph and more heuristic techniques, we can significantly reduce the complexity of the 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 [LP19], 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 100k US$ to compute a chosen-prefix collision. We give more detailed complexity and cost estimates in Table 2.

We have successfully run the computation during two months last summer, using 900 GPUs (Nvidia GTX 1060). Our attack uses one partial block for the birthday stage, and 9 near-collision blocks. We paid 75k US$ to rent the GPUs from GPUserversrental, but actual price could be smaller because we lost some time tuning the attack. There is

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 35 US$/month. To attack `MD5` ‖ `SHA-1`, we use the multicollision attack of Joux [Jou04] with three phase: (i) a CPC on `SHA-1`, (ii) 64 collisions on `SHA-1`, and (iii) 2^{64} evaluations of `MD5`.

Function	Collision type	GPU	Time	Complexity	Cost
`SHA-1`	collision	GTX 970	22 years	$2^{61.2}$	
		GTX 1060	27 years	$2^{61.6}$	11k US$
		GTX 1080 Ti	8 years	$2^{61.6}$	
	chosen-prefix	GTX 970	99 years	$2^{63.4}$	
		GTX 1060	107 years	$2^{63.5}$	45k US$
		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	$2^{67.6}$	720k US$
		GTX 1080 Ti	540 years	$2^{67.6}$	

also a large variability depending on luck, and GPU rental prices fluctuate together with cryptocurrencies 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 the OpenPGP and JPEG format, we can create two public 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 has 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 will still be 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 show that `SHA-1` is now fully and practically broken for use in digital signatures. GPU technology improvements and general computation cost decrease will quickly render our attack even cheaper, making it basically possible 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 will downgrade the connection to `SHA-1`. `SHA-1` is also the foundation of the GIT versioning system, and 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 [Nat95] for all the complete details) and summarize the previous cryptanalysis work relevant to our new findings.

2.1 Description of SHA-1

SHA-1 is a 160-bit hash function that follows the well-known Merkle-Damgård paradigm [Dam89, Mer89]. A padding is first applied to the message input (with message length encoded) so that we obtain a multiple of 512 bits, and this bit string is divided into blocks m_i of 512 bits each. Then, each block m_i is processed via the SHA-1 compression function (denoted h) to update a 160-bit chaining variable (denoted cv_i) that is initialised to a constant and public initial value (denoted IV): $cv_0 = IV$. More precisely, we have $cv_{i+1} = h(cv_i, m_{i+1})$. When all blocks have eventually been processed, the last chaining variable is the final hash output.

The SHA-1 compression function resembles other members of the MD-SHA family of hash functions. It 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 modular addition.

The internal block cipher is composed of 4 rounds of 20 steps each (for a total of 80 steps), where one step follows a generalised Feistel network. More precisely, the internal state is divided into five registers $(A_i, B_i, C_i, D_i, E_i)$ of 32-bit each and at each step, an extended message word W_i updates the registers as follows:

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

where K_i are predetermined constants and f_i are boolean functions (in short: IF function for the first round, XOR for the second and fourth round, MAJ for the third round, see Table 3). Since only a single register value is updated (A_{i+1}), the other registers being only rotated copies, we can express the SHA-1 step function using a single variable:

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

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

Table 3: Boolean functions and constants of `SHA-1`

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

32-bit words $M_0, \ldots, M_{15}$, and then the W_i's 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

Without going into too many details of the very technical `SHA-1` cryptanalysis advances, 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.

2.2.1 Preparing a Collision Attack on `SHA-1`

The first results on `SHA-0` (the predecessor of `SHA-1`) and `SHA-1` were differential in nature and obtained by building differential paths from a linearization of the compression function: in order to simplify the analysis the attacker assumes that modular additions and boolean functions f_i in the `SHA-1` compression function are behaving as an XOR with regards to differential propagation. These assumptions are indeed happening with a certain probability, which will basically consist of the bulk of the final attack cost. Then, in order to further simplify the analysis while trying to minimize the number of differences present in the internal state (which will in turn increase the differential trail probability and therefore improve the final attack cost), these 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, while following the `SHA-1` message expansion. However, in this linearization model, impossibilities might appear in the first 20 steps of `SHA-1` (as in some specific cases the f_{IF} boolean function will never behave as an XOR) and the cheapest trail candidates might not be the ones that start and end with the same difference (which is a property required to obtain directly a 1-block collision after the compression function feed-forward). This strategy could already break the collision resistance of `SHA-0`, but not yet for `SHA-1` (due to a small rotation added in the message expansion of `SHA-1`, that forces disturbances to spread throughout the rounds).

A huge breakthrough then happened in 2005: a team of researchers [WYY05] showed that by generating non-linear differential trails for the first 10∼15 steps of the compression function, one could potentially connect any incoming input difference to any fixed difference δ_I at step 10∼15. This flexibility allows to remove completely the impossibility issues one could face in the first steps due to the linearization (since this part is now non-linear). Even better, taking advantage of the Davies-Meyer construction used inside the compression function, it actually permits to perform the collision attack on `SHA-1` using only two blocks containing differences, while picking the cheapest differential trail from step 10∼15 to 80. With two successive blocks using the same differential trails (just ensuring that the output difference of the two blocks have opposite signs: $0 \overset{\delta_M}{\rightsquigarrow} \delta_O$ and $\delta_O \overset{-\delta_M}{\rightsquigarrow} -\delta_O$), one can see in Figure 1 that a collision is obtained at the end of the second block because the internal state differences cancel out.

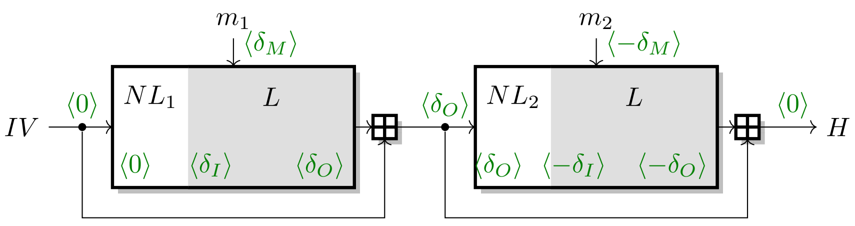


Figure 1: 2-block collision attack using a linear trail $\delta_I \overset{\delta_M}{\rightsquigarrow} \delta_O$ and two non-linear trails $0 \rightsquigarrow \delta_I$ and $\delta_O \rightsquigarrow -\delta_I$ in the first 10 15 steps. Green values between bracket represent differences in the state.

2.2.2 Computing a Collision for SHA-1 with Amortization Techniques

Once the differential trail is set, for each successive block the attacker can concentrate on finding a pair of messages that follows it. For this, he constructs a large number of messages that follow the trail up to some predetermined step (using the freedom degrees available from the message input), and then computes the remaining SHA-1 steps to test whether the output difference is the expected one δ_O (this part is only probabilistic). Compared to a pure naive search, it can be observed that the computational cost is reduced by using a simple early-abort strategy for the 16 first steps. Yet, more advanced amortization methods such as neutral bits [BCJ+05], boomerangs [Kli06, JP07] or message modification [WYY05] can reach a few more steps further. Because of this amortization, usually the first 20 or so steps do not contribute to the final complexity of the attack (which is a good thing because the first 10∼15 steps of the differential path are non-linear and thus will be verified with quite low probability).

Neutral bits were firstly introduced for the cryptanalysis of SHA-0 [BC04, BCJ+05]. The idea is that once a message pair following the differential path until a certain step x was found, one could get another message pair valid until step x for free by applying a small message modification (one or a few bits). The hope is basically that such a modification would not interact too much with necessary conditions in the differential path before step x. The fact that a certain modification was neutral or not until a step x with a good probability could be pre-analysed before running the attack and a key observation was that any combination of two of more neutral bits until step x was likely to be neutral as well until step x, which gives an exponential potential amortization.

Boomerangs [JP07] or tunnels [Kli06] are very similar amortization tools to neutral bits. Basically, they can be seen as neutral bits that are planned in advance: build from one or a few local collisions (or relaxed versions of a local collision), one expects this perturbation to be 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 (in the sense that they produce candidates valid with high probability for later steps than classical neutral bits), but consume more freedom degrees. For this reason, you can generally only place a few of them due to a lack of available freedom degrees, 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 some internal state bits and potentially message bits must be fulfilled in order for the planned differential path to be valid. Thus, one can't place neutral bits or boomerangs anywhere, but only at very particular bit positions and steps.

2.2.3 Chosen-prefix Collision Attacks

Chosen-prefix collision attacks are difficult to find for iterated hash functions such as SHA-1, because the attacker's task is then to find a collision while starting from a random difference in the internal state (due to the prefixes pair that is not controlled at all by him). This random difference prevents to use directly the collision search techniques for SHA-1 aforementioned, because the attacker has to erase this random difference somehow and the interesting differential paths are in fact a very small set, where all output differences δ_O only have a very low Hamming weight.

The first concrete application of a chosen-prefix collision attack was on MD5 [SLdW07] and this work was also the first to introduce a birthday search phase in order to partially avoid the random difference issue. The idea is to process random message blocks after the challenged prefixes, until the chaining variable difference δ belongs to a large predetermined set $\mathcal{S}$. Since the message blocks after each prefix are chosen independently, this can be done with birthday complexity $\sqrt{\pi \cdot 2^n/|\mathcal{S}|}$. Then, from that difference δ, one can reach a collision by slowly erasing the remaining unwanted difference bits by successfully applying some near-collision blocks (see Figure 2 with the example of a unique near-collision block). We note that the starting difference set $\mathcal{S}$ during the birthday phase must not be too small, otherwise this phase would be too costly. Moreover, the near-collisions blocks must not be too expensive either, and this will of course depend on the cryptanalysis advancements of the compression function being studied. This two-phase strategy was applied in [Ste13b] to the full SHA-1, 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 generate enough allowable differences that can later be erased efficiently with a near-collisions block, which makes the birthday part by far the most expensive phase of the attack. In [Ste13b] a set $\mathcal{S}$ of 192 allowable differences was used, by starting from one type of near collision block and then varying the signs of the message and output differences, and also by letting some uncontrolled differences spread during the very last steps of the compression function.

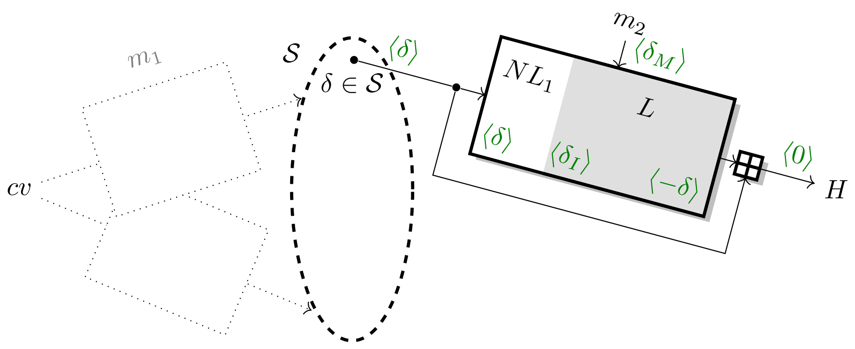


Figure 2: Single-block chosen-prefix collision attack with a birthday stage. The linear trail $\delta_I \rightsquigarrow \delta_O$ is relaxed to reach a set $\mathcal{S}$ of feasible differences.

This was improved in [LP19] by generalising for SHA-1 the set of possible differences that can be obtained for a cheap cost with a single message block, increasing the set size to 8768 elements. Another crucial improvement from [LP19] is the utilization of a multi-block strategy for SHA-1 that allows to further greatly increase the size of the set $\mathcal{S}$: the idea is that if an arbitrary input difference δ_R can be decomposed as $\delta_R = -\big(\delta_O^{(1)} + \delta_O^{(2)} + \cdots + \delta_O^{(r)}\big)$, where each $\delta_O^{(i)}$ can be reached as the output of a differential trail, the attacker just has to find near-collision blocks with output differences $\delta_O^{(1)}, \dots, \delta_O^{(r)}$ (see Figure 3). 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 blocks, 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.

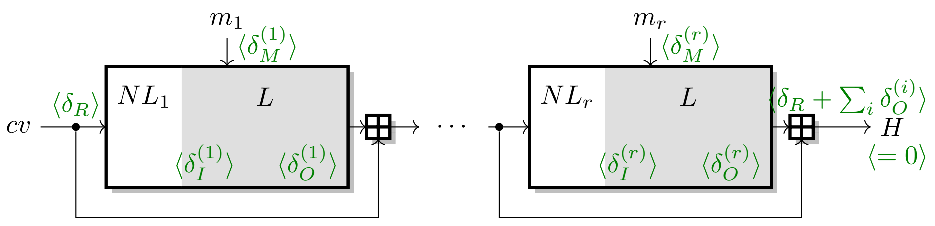

m_1
$\langle\delta_M^{(1)}\rangle$
m_r
$\langle\delta_M^{(r)}\rangle$
cv
$\langle\delta_R\rangle$
NL_1
L
$\langle\delta_I^{(1)}\rangle$
$\langle\delta_O^{(1)}\rangle$
$\cdots$
NL_r
L
$\langle\delta_I^{(r)}\rangle$
$\langle\delta_O^{(r)}\rangle$
$\langle\delta_R + \sum_i \delta_O^{(i)}\rangle$
H
$\langle = 0\rangle$

Figure 3: Multi-block chosen-prefix collision attack. We assume that an arbitrary difference δ_R can be decomposed as $\delta_R = -\left(\delta_O^{(1)} + \delta_O^{(2)} + \cdots + \delta_O^{(r)}\right)$, where each $\delta_O^{(i)}$ can be reached as the output of a differential trail.

3 SHA-1 Collision Attack Improvements

While the work of [LP19] 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 the near-collision search.

3.1 Analysis of Previous Boomerangs and Neutral Bits

As explained in [LP19], an important factor to evaluate the cost of the attack is the number of boomerangs available when looking for a conforming message. The collision attack from Eurocrypt 2013 [Ste13b] and its GPU implementation from Crypto 2017 [SBK+17] use three boomerangs, on bits 6 and 8 of M_6 (red type in Figure 4), and on bit 7 of M_9 (blue type in Figure 4). However, the boomerang on $M_6[8]$ actually flips the value of $W_{77}[0]$ due to the message expansion, and break the condition $W_{77}[0] \oplus W_{77}[2] = 1$ listed on Table 5 of [SBK+17]. This reduces the probability of the trail on rounds 61—80 by a factor $3/4$, from $2^{-19.17}$ to $2^{-19.58}$. However, the boomerang almost doubles the number of partial solution produced, so that using it still improves the attack.

In addition, we realized that the attack implemented by Stevens *et al.* uses a neutral bit on $M_{13}[11]$. However, this breaks the condition $W_{76}[0] \oplus W_{76}[1] = 1$, and reduces the probability of the trail by a factor roughly $2^{0.2}$. In our analysis, we assume that this neutral bit has been removed: since it has a very small effect on the number of partial solution produced, it reduces the complexity of the attack by a factor $2^{0.2}$. In particular, this is why we consider that a collision requires $2^{48.5}$ A_{33}-solutions rather than $2^{48.7}$.

In our chosen-prefix attack, we also need conditions on $W_{77}[0]$, and we decided to remove the boomerang on $M_6[8]$ in order to keep more control on the output difference and to simplify the attack. In particular, removing this boomerang makes easier the construction of trails with the extra constraints (and we have successfully built trails with the two remaining boomerangs for all successive blocks). This implies that the cost of near-collision blocks increases by a factor $3/2$ compared to the shattered attack, leading to $C_{\text{block}} = 2^{64.9}$ on a GTX 970 (after gaining a factor $2^{0.2}$ by removing $M_{13}[11]$ as a neutral bit). Therefore we can estimate more accurately the complexity of the previous

i	A_i	W_i
-1:	--------------------------------	
00:	--------------------------------	--------------------------------
01:	--------------------------------	--------------------------------
02:	--------------------------------	--------------------------------
03:	--------------------------------	--------------------------------
04:	--------------------------------	--------------------------------
05:	---------------------------|----	--------------------------------
06:	---------------------------|----	-----------------------------x--
07:	-----------------------------x--	------------------------$\bar{x}$-------
08:	---------------|---------------0	--------------------------------
09:	---------------|---------------1	-----------------y--------------
10:	----|------------y--------------	------------$\bar{y}$-------------------
11:	----|--------------0------------	------z------------------------$\bar{x}$
12:	------z------------1------------	-$\bar{z}$------------------------------
13:	--------0-----------------------	--------------------------------
14:	--------1-----------------------	-------------------$\bar{y}$------------
15:	--------------------------------	--------------------------------

Figure 4: Boomerangs differential paths used for SHA-1 with the corresponding constraints forced in order to have probability one in the 16 first steps. The red one (perturbation x) represents a small boomerang (named AP_1 in [MP08]) composed of a single local collision starting on M_6, here positioned at bit $j = 2$. The blue one (perturbation y) represents another small boomerang used in [Ste13b, SBK$^+$17], also composed of a single local collision, but starting on M_9, here positioned at bit $j = 14$. The green one (perturbation z) represents a new and even smaller boomerang built from a partial local collision starting on M_{11}, here positioned at bit $j = 25$. The MSB's are on the right and "-" stands for no constraint. The letters represent a bit value and its complement is denoted by an upper bar on the corresponding letter. The notation "|" on two bits vertically neighbour mean that these two bits must be equal.

i	A_i	W_i
-1:	--------------------------------	
00:	--------------------------------	--------------------------------
01:	--------------------------------	--------------------------------
02:	--------------------------------	--------------------------------
03:	--------------------------------	--------------------------------
04:	--------------------------------	--------------------------------
05:	---------------------|-|--------	--------------------------------
06:	---------------------|-|--------	-----------------------0-0------
07:	-----------------------0-0------	------------------1-1-----------
08:	----------------------|--0-0----	--------------------------------
09:	----------------------|--1-1----	------------------------0-------
10:	-----------------------|0|------	-------------------1------------
11:	-----------------------|0|0-----	-------------------------111----
12:	-------------------------111----	--------------------------------
13:	---------------------------000--	--------------------------1-----
14:	---------------------------111--	--------------------------1-----
15:	--------------------------------	--------------------------------

Figure 5: Exact conditions required to prepare all the boomerangs differential paths used for our CP collision attack on SHA-1 with the corresponding constraints forced in order to have probability one in the 16 first steps. The MSB's are on the right and "-" stands for no constraint. The notation "|" on two bits vertically neighbour mean that these two bits must be equal. These conditions basically correspond to very short boomerangs started at $M_{11}[4]$, $M_{11}[5]$ and $M_{11}[6]$, and small boomerangs started at $M_6[6]$, $M_6[8]$ and $M_9[7]$ (boomerang starting perturbations are marked in purple). We remark an extra condition $M_{13}[5] = 1$ in order to potentially correct the clash between boomerangs $M_{11}[5]$ and $M_9[7]$.

attack [LP19] as $2^{67.1}$ `SHA-1` computations, instead of the range of $2^{66.9}$ to $2^{69.4}$ reported previously (the optimal attack parameter choice for [LP19] is then a maximum cost of 3.5 C_{block}).

As an optimization, we decided to use the boomerang on $M_6[8]$ for the last block, because the computation of the last block is identical to the second block of an identical-prefix collision attack. In particular, we had no trouble building a path with this boomerang, and this results in a speed-up of a factor 1.9 in the rate of A_{33}-solutions. We need to increase the number of solutions by a factor $4/3$ because the solutions are of lower quality, but this still corresponds to a speed-up factor of roughly 1.4. Since the last block represents a significant part of the total computation, this is a worthy optimization.

3.2 Improvements to `SHA-1` Near-collision Search

When looking at low-level details of the near-collision search, we found several ways to improve the near-collision search of the shattered attack [SBK+17]. 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.

Extra boomerangs and modular correction for boomerangs. We found out that in addition to previously mentioned boomerangs, we can use very short boomerangs on bits 4, 5, and 6 of M_{11} (green type in Figure 4), with a single correction on M_{12}. These boomerangs are neutral until step 22, like the boomerang on M_9. The problem is that they will clash with existing small boomerangs starting at $M_6[6]$, $M_6[8]$ and $M_9[7]$.

More precisely, the boomerang starting at $M_{11}[4]$ will flip the message condition "$\overline{\texttt{x}}$" from the boomerang starting at $M_6[6]$, corresponding to the last message correction of the local collision. In order to avoid this issue, we simply change the last correction of the $M_6[6]$ boomerang to be a modular addition correction instead of an XOR correction. This will naturally correct the perturbation as the step operation involved is indeed a modular addition and the boomerang on $M_6[6]$ will behave as expected (the condition "$\overline{\texttt{x}}$" is actually not needed anymore). This can be seen as a generalization of the boomerang strategy used so far for `SHA-1`: boomerang corrections can be applied modular addition-wise instead of XOR-wise. This will induce changes in subsequent bits in the corresponding message words because of carry propagations that might naturally occur with a modular addition operation, but as long as these bit changes do not mess with existing message conditions, we are fine[1]. This idea might also be interesting to analyse other hash functions.

Similarly, the boomerang starting at $M_{11}[6]$ will flip the message condition "$\overline{\texttt{x}}$" from the boomerang starting at $M_6[8]$, corresponding to the last message correction of the local collision. This is exactly the same situation and we avoid this issue by changing the last correction of the $M_6[8]$ boomerang to be a modular addition correction instead of an XOR correction.

Finally, the boomerang starting at $M_{11}[5]$ will flip the condition "`1`" in the internal state, required for the boomerang starting at $M_9[7]$. This will create an uncontrolled difference when we will use the $M_9[7]$ boomerang, that we then correct by introducing a new difference in $M_{13}[5]$ (setting the extra condition $M_{13}[5] = M_9[7] \oplus 1$ will ensure a proper correction using XOR correction). Note that we perform this further correction only in the case where the boomerang starting at $M_{11}[5]$ was triggered. We observe furthermore that the correction on $M_{13}[5]$ will maintain a good quality of neutralness for $M_{11}[5]$ boomerang.

All the conditions required to use the boomerangs are given in Figure 5.

[1] In more details, since the boomerangs on $M_{11}[4]$, $M_{11}[5]$ and $M_{11}[6]$ are very short and thus applied before boomerangs $M_6[6]$, $M_6[8]$ and $M_9[7]$, we don't actually care if the last ones break conditions of the first ones since they have already been used.

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` is overestimated).

			Collision (old)		Collision (new)		
GPU	arch	Hashrate	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

Neutral bits/message modifications: a better use of degrees of freedom. We also improved the rate of A_{33}-solutions generated by looking into more details at the effect of each neutral bit. In particular, we found that some neutral bits will flip with very high probability a certain condition at a later step than their neutralness step. Therefore, these neutral bits can potentially be used as message modification bits 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 bit instead of neutral bit will be 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 any 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 [SBK+17] (including the boomerang on $M_6[8]$). If we remove the boomerang on $M_6[8]$, this results in $C_{\text{block}} = 2^{61.7}$ on a GTX 970.

3.3 Building Differential Trails

Following [LP19], 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.* [SBK+17], 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 [SBK+17]: 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 use boomerangs and neutral bits as explained in previous subsection.

The starting path skeleton is depicted in Figure 6. 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;

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 6: 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 neighbour mean that these two bits must be equal. The other notations are similar to the ones used in [DR06]. 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.

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 that consist of 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.

Step 1 to 4 are done on CPU because they are not too computationally intensive, but step 5 is implemented on GPU.

We observe that in comparison with the classical collision attack [SBK$^+$17], we have fewer degrees of freedom available in our differential paths, due to slightly more linear constraints we imposed on the late-step message bits. Moreover, for the first blocks of the near-collision phase, the attacker will have to handle a denser input difference on the chaining variable, which will render the non-linear part search a little more difficult and little more consuming in terms of freedom degrees. In any cases, we had enough degrees of freedom to find a conforming messages pair for all blocks during the attack.

We also remark that the incorporation of the extra short boomerangs reduces the number of neutral bits that can be used in comparison to [SBK$^+$17]. Yet, this was not an issue as 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 SHA-1 Chosen-Prefix Collision Attack Improvement

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. In particular, we need a larger set $\mathcal{S}$ to reduce the complexity of the birthday phase (in [LP19], the largest graph suggested has size $2^{33.7}$, corresponding to a birthday cost of $2^{64.3}$). We also use a more heuristic approach than in [LP19], resulting a lower average complexity, but without a guaranteed upper bound on the complexity.

4.1 Larger Graph

We started with the same approach as in [LP19], building a series of graphs with increasing limits on the number of blocks allowed, but we improved the code (in particular the memory usage) in order to reach higher sizes. More precisely, we start with the set of all nodes that are reachable with a path of cost at most 24 C_{block} and up to 10 blocks, 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 3TB of RAM, we have obtained a graph with $2^{36.2}$ nodes, which requires 2TB of storage (storing only the nodes and their cost). While we initially consider nodes at distance up to 24, after the clustering we find that almost 90% of the nodes are actually at distance 6 or less, as seen in Table 5.

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

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

We found that 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 to see if they reach the graph and compute the cost using the clustering. This corresponds to a bi-directional search were we pre-compute in the backwards direction the set a values that

[2]Given by mask 0x7f000000, 0xfff80001, 0x7ffff000, 0x7fffffc0, 0x7fffffff

go to zero after at most 10 blocks, and during the online phase, we compute one block forward.

This can be seen as a time-memory trade-off: we can now use nodes at a distance up to 11 blocks, but we only build explicitly the graph with 10 blocks. Moreover, we can now 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 cannot 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 [LP19]).

4.3 More Dynamic Blocks

Following [LP19], 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 collide at 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.

Again, we cannot 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 block that expected, in particular for the first near-collision blocks.

5 `SHA-1` 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 well within an academic budget, for a total cost much lower than 100k US$.

5.1 Attack Parameters

Using more blocks for the near-collision blocks part of the attack leads to improved attack complexity, but one can observe that the improvement becomes marginal as the number of block increases. Besides, in order to allow for a practical real-life use of our chosen-prefix collision (see next section), we had to enforce a limit on the total number of blocks.

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 uses 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 of length 2^{28}, resulting in a data complexity of $1/2$ TB to store 2^{34} chains;
- We expect a cost of $2\ C_{\text{block}}$ for the near-collision phase.

In hindsight, we could have use longer chains to reduce the data storage, and to make sorting the data on our cluster easier. We could also have aimed for a lower number of blocks: our collision used only 9 blocks, and we could probably reach 8 or 7 without much impact on the complexity.

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 Section 3.1, 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 our identical-prefix collision attack.

5.2 A GPU Cluster

We originally estimated that our attack would cost around 160k US$ 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 75k US$ to compute a chosen-prefix collision.

After some cost analysis, we have concluded that GTX 1060 GPUs offered a very good hashrate/cost ratio at the time of the chosen-prefix collision computation. GPU prices vary significantly depending on cryptocurrency prices, but at the time of writing, a GTX 1060 can be rented for about 35 US$ per month[3]. Our attack requires about 107 GPU-years using GTX 1060, which gives an estimated cost of $107 \times 35 \times 12 \simeq 45$k US$ to compute a chosen-prefix collision for `SHA-1`.

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

[3]More precisely, 209 US$ per month for 6 GTX 1060 3GB: `https://web.archive.org/web/20191229164814/https://www.gpuserversrental.com/`

Table 6: Timeline of the birthday phase

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!	$2^{62.9}$	$2^{27.7}$

Pentium or Celeron CPU with two cores. Each machine ran Ubuntu Linux, but there was no cluster management software installed (we used cluster shell to run commands on all the nodes). We negotiated a price of 37.8k US$ per month (which is actually higher than current prices), and used the cluster for two months.

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

This cryptanalysis was of course always possible, yet not public (and perhaps not yet discovered). Therefore, we can try to estimate how much it would have cost in the past. In 2009, Stevens *et al.* used a cluster of 215 PS3 gaming consoles to attack MD5 [SSA+09], with a computation power of roughly 40GH/s for MD5. If we assume that this cluster could compute SHA-1 with a rate of 20GH/s and that the attack cost would be similar at $2^{63.5}$ SHA-1, it would have taken 20 years to compute a chosen-prefix collision for SHA-1 on that cluster. In 2010 the US Air Force built a cluster out of 1760 PS3 for a cost of 2 million US$[4]. Using our estimation, this cluster would have taken 2 and a half years to compute a SHA-1 chosen-prefix collision, so we can estimate that the cost of such an attack would have been a few million US$[5] in 2010, when SHA-1 was still the most widely used hash function.

Looking at the future, it is clear that this chosen-prefix collision attack will get even cheaper as computation costs decrease. Using Moore's law estimation (that seems to be still valid for GPU[6]), we evaluate that it should cost less than 10k US$ to generate a chosen-prefix collision for SHA-1 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 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 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

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

[5] Assuming a unit price of 400 US$ and a power draw of 130W at 10 cent/kWh, the cost would be 1.2 million US$

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

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 $2^{19.58}$

Date	Event	# A_{61}-solutions	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}

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

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).

Interestingly, we got slightly fewer collisions than expected (after a given number of chains): we expected to compute $\sqrt{\pi 2^{96} C}$ `SHA-1` to find C partial collisions, but our analysis is off by a factor roughly $2^{0.2}$. Given the small magnitude of the error, we didn't investigate further, but it could be due to an unknown bug in our code, or an issue in the analysis (such as a failed independence assumption, or an issue with chains that reach a cycle).

A timeline of the birthday phase is given in Table 6.

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, in order to connect the new incoming chaining variable to the core path. We used tools similar to [DR06], 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. Finally, some testing and configuration of the GPU code was then required to check how neutral bits and boomerangs behaved in this new configuration. In particular, there were usually some adjustments to make in the GPU code for the more complex conditions in the path that involve several bit positions. The entire preparation process would have potentially to be performed again in case of any issue detected with the path found.

This was automated to some extend, 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 costed even cheaper if we had fully automatized the entire cryptanalysis process, or if we improved the non-linear part search algorithm. This is definitely not impossible to achieve, but it would require a lot a tedious

Table 8: Resources used for the attack

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 $\times$	$\approx$ 1 day
	Locating collisions	GPU	4 $\times$	$<$ 1/2 day
	Searching in graph	RAM	4 $\times$	$<$ 1/2 day
Blocks	Building trail & code	Human Time	9 $\times$	$\approx$ 1 day
	Finding block	GPU	8 $\times$	3 hours – 3 days
	Checking results in graph	RAM	8 $\times$	$<$ 1/2 hour
	Finding last block	GPU	1 $\times$	6 days

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 time 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 have been 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 full chosen-prefix collision is given in Figure 7.

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
- 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. This trust model relies on users signing each other's identity certificate, instead of using a central PKI. For compatibility reasons the legacy branch of GnuPG (version 1.4) still uses `SHA-1` by default to sign identity certificates.

Therefore, we can impersonate a user using a `SHA-1` chosen-prefix collision to forge the signature. 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

	Message A	Message B
0x0000	99 04 0d 04 7f e8 17 80 01 20 00 ff 4b 65 79 20	99 03 0d 04 7f e8 17 80 01 18 00 ff 50 72 61 63
	69 73 20 70 61 72 74 20 6f 66 20 61 20 63 6f 6c	74 69 63 61 6c 20 53 48 41 2d 31 20 63 68 6f 73
	6c 69 73 69 6f 6e 21 20 49 74 27 73 20 61 20 74	65 6e 2d 70 72 65 66 69 78 20 63 6f 6c 6c 69 73
	72 61 70 21 79 c6 1a f0 af cc 05 45 15 d9 27 4e	69 6f 6e 21 1d 27 6c 6b a6 61 e1 04 0e 1f 7d 76
0x0040	73 07 62 4b 1d c7 fb 23 98 8b b8 de 8b 57 5d ba	7f 07 62 49 dd c7 fb 33 2c 8b b8 c2 b7 57 5d be
	7b 9e ab 31 c1 67 4b 6d 97 43 78 a8 27 73 2f f5	c7 9e ab 2b e1 67 4b 7d b3 43 78 b4 cb 73 2f e1
	85 1c 76 a2 e6 07 72 b5 a4 7c e1 ea c4 0b b9 93	89 1c 76 a0 26 07 72 a5 10 7c e1 f6 e8 0b b9 97
	c1 2d 8c 70 e2 4a 4f 8d 5f cd ed c1 b3 2c 9c f1	7d 2d 8c 68 52 4a 4f 9d 5f cd ed cd 0b 2c 9c e1
0x0080	9e 31 af 24 29 75 9d 42 e4 df db 31 71 9f 58 76	92 31 af 26 e9 75 9d 52 50 df db 2d 4d 9f 58 72
	23 ee 55 29 39 b6 dc dc 45 9f ca 53 55 3b 70 f8	9f ee 55 33 19 b6 dc cc 61 9f ca 4f b9 3b 70 ec
	7e de 30 a2 47 ea 3a f6 c7 59 a2 f2 0b 32 0d 76	72 de 30 a0 87 ea 3a e6 73 59 a2 ee 27 32 0d 72
	0d b6 4f f4 79 08 4f d3 cc b3 cd d4 83 62 d9 6a	b1 b6 4f ec c9 08 4f c3 cc b3 cd d8 3b 62 d9 7a
0x00c0	9c 43 06 17 ca ff 6c 36 c6 37 e5 3f de 28 41 7f	90 43 06 15 0a ff 6c 26 72 37 e5 23 e2 28 41 7b
	62 6f ec 54 ed 79 43 a4 6e 5f 57 30 f2 bb 38 fb	de 6f ec 4e cd 79 43 b4 4a 5f 57 2c 1e bb 38 ef
	1d f6 e0 09 00 10 d0 0e 24 ad 78 bf 92 64 19 93	11 f6 e0 0b c0 10 d0 1e 90 ad 78 a3 be 64 19 97
	60 8e 8d 15 8a 78 9f 34 c4 6f e1 e6 02 7f 35 a4	dc 8e 8d 0d 3a 78 9f 24 c4 6f e1 ea ba 7f 35 b4
0x0100	cb fb 82 70 76 c5 0e ca 0e 8b 7c ca 69 bb 2c 2b	c7 fb 82 72 b6 c5 0e da ba 8b 7c d6 55 bb 2c 2f
	79 02 59 f9 bf 95 70 dd 8d 44 37 a3 11 5f af f7	c5 02 59 e3 9f 95 70 cd a9 44 37 bf fd 5f af e3
	c3 ca c0 9a d2 52 66 05 5c 27 10 47 55 17 8e ae	cf ca c0 98 12 52 66 15 e8 27 10 5b 79 17 8e aa
	ff 82 5a 2c aa 2a cf b5 de 64 ce 76 41 dc 59 a5	43 82 5a 34 1a 2a cf a5 de 64 ce 7a f9 dc 59 b5
0x0140	41 a9 fc 9c 75 67 56 e2 e2 3d c7 13 c8 c2 4c 97	4d a9 fc 9e b5 67 56 f2 56 3d c7 0f f4 c2 4c 93
	90 aa 6b 0e 38 a7 f5 5f 14 45 2a 1c a2 85 0d dd	2c aa 6b 14 18 a7 f5 4f 30 45 2a 00 4e 85 0d c9
	95 62 fd 9a 18 ad 42 49 6a a9 70 08 f7 46 72 f6	99 62 fd 98 d8 ad 42 59 de a9 70 14 db 46 72 f2
	8e f4 61 eb 88 b0 99 33 d6 26 b4 f9 18 74 9c c0	32 f4 61 f3 38 b0 99 23 d6 26 b4 f5 a0 74 9c d0
0x0180	27 fd dd 6c 42 5f c4 21 68 35 d0 13 4d 15 28 5b	2b fd dd 6e 82 5f c4 31 dc 35 d0 0f 71 15 28 5f
	ab 2c b7 84 a4 f7 cb b4 fb 51 4d 4b f0 f6 23 7c	17 2c b7 9e 84 f7 cb a4 df 51 4d 57 1c f6 23 68
	f0 0a 9e 9f 13 2b 9a 06 6e 6f d1 7f 6c 42 98 74	fc 0a 9e 9d d3 2b 9a 16 da 6f d1 63 40 42 98 70
	78 58 6f f6 51 af 96 74 7f b4 26 b9 87 2b 9a 88	c4 58 6f ee e1 af 96 64 7f b4 26 b5 3f 2b 9a 98
0x01c0	e4 06 3f 59 bb 33 4c c0 06 50 f8 3a 80 c4 27 51	e8 06 3f 5b 7b 33 4c d0 b2 50 f8 26 bc c4 27 55
	b7 19 74 d3 00 fc 28 19 a2 e8 f1 e3 2c 1b 51 cb	0b 19 74 c9 20 fc 28 09 86 e8 f1 ff c0 1b 51 df
	18 e6 bf c4 db 9b ae f6 75 d4 aa f5 b1 57 4a 04	14 e6 bf c6 1b 9b ae e6 c1 d4 aa e9 9d 57 4a 00
	7f 8f 6d d2 ec 15 3a 93 41 22 93 97 4d 92 8f 88	c3 8f 6d ca 5c 15 3a 83 41 22 93 9b f5 92 8f 98
0x0200	ce d9 36 3c fe f9 7c e2 e7 42 bf 34 c9 6b 8e f3	c2 d9 36 3e 3e f9 7c f2 53 42 bf 28 f5 6b 8e f7
	87 56 76 fe a5 cc a8 e5 f7 de a0 ba b2 41 3d 4d	3b 56 76 e4 85 cc a8 f5 d3 de a0 a6 5e 41 3d 59
	e0 0e e7 1e e0 1f 16 2b db 6d 1e af d9 25 e6 ae	ec 0e e7 1c 20 1f 16 3b 6f 6d 1e b3 f5 25 e6 aa
	ba ae 6a 35 4e f1 7c f2 05 a4 04 fb db 12 fc 45	06 ae 6a 2d fe f1 7c e2 05 a4 04 f7 63 12 fc 55
0x0240	4d 41 fd d9 5c f2 45 96 64 a2 ad 03 2d 1d a6 0a	41 41 fd db 9c f2 45 86 d0 a2 ad 1f 11 1d a6 0e
	73 26 40 75 d7 f1 e0 d6 c1 40 3a e7 a0 d8 61 df	cf 26 40 6f f7 f1 e0 c6 e5 40 3a fb 4c d8 61 cb
	3f e5 70 71 88 dd 5e 07 d1 58 9b 9f 8b 66 30 55	33 e5 70 73 48 dd 5e 17 65 58 9b 83 a7 66 30 51
	3f 8f c3 52 b3 e0 c2 7d a8 0b dd ba 4c 64 02 0d	83 8f c3 4a 03 e0 c2 6d a8 0b dd b6 f4 64 02 1d

Figure 7: Chosen-prefix collision for `SHA-1`. The colors show the prefix, the birthday bits, and the near-collision blocks.
Both messages have the same `SHA-1`: `8ac60ba76f1999a1ab70223f225aefdc78d4ddc0`

Table 9: Differences in the state after each block, and output differences at the end of the trail (before the feed-forward)

Block	State difference				
Birthday	-------------n----n---u--n-u-n-u	------------------n----n-u-n-u--	u------------------------n------	-----------------------------n-n	u-------------------------------
Block 1	u------------n-u-n--u---u-u--u--	------------------n-u-n-u--n-n--	-------------------------n-u----	u---------------------------n-u-	--------------------------------
Block 2	-----------------n--u------n-u--	----------------------n-u-n--u--	-------------------------------u	----------------------------n-u-	u-------------------------------
Block 3	------------n----n-n--n--n--n-n-	-----------------n----n-n-----u-	------------------------n-------	----------------------------n---	--------------------------------
Block 4	------------n-u--n--u-n-n-n-u--u	-----------------n-u--n--u---n--	u-----------------------n-u----u	----------------------------n--n	u-------------------------------
Block 5	------------n----n-u-u----u-n-n-	-----------------n-----n-n-n--n-	u-----------------------n------u	----------------------------n---	--------------------------------
Block 6	------------------n-n----n-n----	-----------------------n-n---n--	u-------------------------------	----------------------------n-u-	u-------------------------------
Block 7	------------------n---n-n-u--n--	-----------------------n---n-n--	--------------------------------	-----------------------------n--	--------------------------------
Block 8	-------------------n--n--n-u--n-	------------------------n--n--n-	--------------------------------	------------------------------n-	u-------------------------------
Block 9	--------------------------------	--------------------------------	--------------------------------	--------------------------------	--------------------------------

Block	Output difference				
Block 1	u--------------u--n-u--n--n-n--n	--------------------u--n-u--n---	u--------------------------u----	u------------------------------n	u-------------------------------
Block 2	u------------u-n-------n-u-u----	------------------u-n-------n---	-------------------------u-n---u	u-------------------------------	u-------------------------------
Block 3	------------n-----n-u-n--n----u-	-----------------n-----n--u---n-	------------------------n------n	------------------------------n-	u-------------------------------
Block 4	--------------u---u-n----n-n-u-n	-------------------u---u-n--n-u-	u-------------------------u----u	-------------------------------n	u-------------------------------
Block 5	--------------n----u---n-n-n-n-u	-------------------n----u--n--u-	--------------------------n-----	-------------------------------u	u-------------------------------
Block 6	------------u--------u--n-u-n-u-	-----------------u---------u--n-	------------------------u------n	------------------------------u-	u-------------------------------
Block 7	--------------------u-n----n-n--	-------------------------u-n----	u-------------------------------	------------------------------u-	u-------------------------------
Block 8	-------------------u-----u-n--u-	------------------------u-----u-	--------------------------------	------------------------------u-	u-------------------------------
Block 9	-------------------u--u--u-n--u-	------------------------u--u--u-	--------------------------------	------------------------------u-	u-------------------------------

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 [SLdW07, SSA+09]. However, 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`.

We recall that a chosen-prefix collision attack works as follows: given two arbitrary prefixes P and P', an attacker can generate two messages M and M' such that $H(P \| M) = H(P' \| M')$. Note that in classical iterated hash functions such as `SHA-1`, given an arbitrary suffix X, we still have $H(P \| M \| X) = H(P' \| M' \| X)$.

6.1 Exploiting a Chosen-prefix Collision

We now focus on the identity certificates that will be hashed and signed. Following RFC 4880 [CDF+07], the hash function 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 a 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 a 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 Hashed Messages in the 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[7] (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 the "`!!`" bytes in key B to make a valid modulus

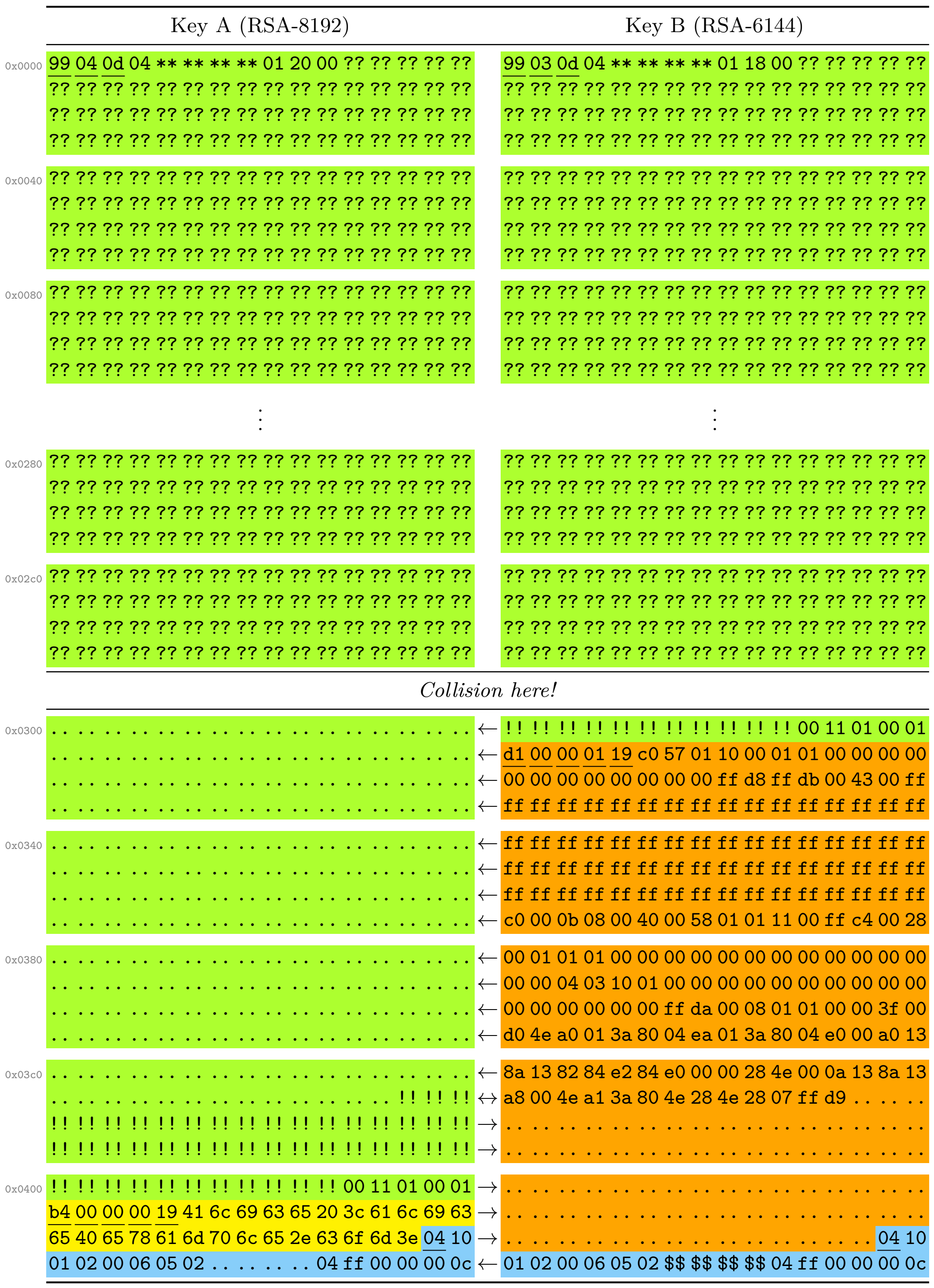
4. Select the "`!!`" bytes in key A to make a valid modulus

5. Generate key B with the modulus and the padded JPEG. Ask for a signature of the key.

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 [SLdW07, SSA+09], where the identifier is hashed before the public key.

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



```
                            Key A (RSA-8192)                                     Key B (RSA-6144)
0x0000 99 04 0d 04 ** ** ** ** 01 20 00 ?? ?? ?? ?? ??     99 03 0d 04 ** ** ** ** 01 18 00 ?? ?? ?? ?? ??
       ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ??     ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ??
       ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ??     ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ??
       ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ??     ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ??

0x0040 ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ??     ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ??
       ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ??     ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ??
       ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ??     ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ??
       ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ??     ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ??

0x0080 ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ??     ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ??
       ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ??     ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ??
       ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ??     ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ??
       ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ??     ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ??

                          ⋮                                                  ⋮

0x0280 ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ??     ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ??
       ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ??     ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ??
       ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ??     ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ??
       ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ??     ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ??

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.

6.1.3 Example Keys

We show an example of a pair of keys generated with this procedure in Figures 9 and 10 from the Appendix. The keys can be examined with `pgpdump -i` to see that they include the same signature. The files 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`

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.

Other variants might also be possible.

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 50k US$, this is well within reach of strong adversaries.

7 Current Usage of `SHA-1` and Responsible 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 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 [BL16] 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 for 15 years, and there are better alternatives available, well-studied, and standardized (`SHA-2` [Nat02], `SHA-3` [Nat15]). 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 [SS17] 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 response 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.

We have first discussed this attack with the GnuPG developers the 9th of May 2019 and eventually informed them of the newly found chosen-prefix collision the 1st of October 2019. The issue is tracked with CVE number CVE-2019-14855. A countermeasure has been implemented in commit `edc36f5`, included in GnuPG version 2.2.18 (released on the 25th of November 2019): `SHA-1`-based identity signatures created after 2019-01-19 are now considered invalid.

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.

However the Web of Trust does not seem to be widely used anymore. In particular, after the poisoning attack at the end of June 2019 [Han19], GnuPG 2.2.17 and later do not import identity certificates from public keyservers by default. A major usage of GnuPG is now to authenticate software packages in Linux, but this typically relies on directly trusting the relevant keys without third parties.

CAcert. CAcert (http://cacert.org/) is one of the main CAs for PGP keys. We noticed that there is a large number of keys with recent `SHA-1` signatures from CAcert on public keyservers. This seems to indicate that 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.

We note that our attack is not directly applicable because CAcert does not sign JPEG images in PGP keys, but using `SHA-1` signature is nonetheless an important security risk.

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.[8] There are also a few old `SHA-1` certificates still in use[9]. 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. Similarly, OpenSSL still accepts `SHA-1` certificates at security level 1 (the default level in most distributions – but Debian Buster has set the default level to 2, which prevents usage of `SHA-1` certificates).

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.*[SSA+09], and in the wild by the flame malware[Ste13a]. If some of the CAs still issuing `SHA-1` certificates use predictable serial numbers, a similar attack might be possible today.

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. In order to authenticate the TLS handshake, the client and the server sign a copy of the transcript at the end of the handshake. If the hash function used in the signature is weak, an attacker can use chosen-prefix collisions to mount a man-in-the-middle attack and break various properties of the handshake, as shown by the SLOTH attacks [BL16]. However, this remains far from being a practical attack, because the CP collision has to be computed in a very short time frame, while the session is being established.

In TLS version 1.0 and 1.1, the handshake is hashed with the concatenation of `SHA-1` and `MD5`. Using the multicollision attack from Joux [Jou04], 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 version 1.2, the hash function used is configurable, and is negotiated between the client and the server. `MD5` was one of the possible options, but support has been removed after the SLOTH attack. However, `SHA-1` is still widely supported, and many servers actually *prefer* to use `SHA-1`, even when the client offers better algorithms. Scan results of the top 1M websites show that 3% of them use `SHA-1`[10], and this includes many

[8] 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`

[9] 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`

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

high profile websites.[11] The vast majority of TLS 1.0/1.1 clients offer `SHA-1` as an option for the signature.

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

Ciphersuites. The ciphersuite used in a TLS connection is the result of a negotiation between the client and server, so it is hard to predict exactly. However, 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. Telemetry results from Mozilla report about 2% of connections with a `HMAC-SHA-1` ciphersuite.[12] In addition, a scan of websites in the Alexa top 1M show that 8% of them would use a `HMAC-SHA-1` ciphersuite with the client settings used for the scan[13].

This usage is not threatened by our attack, but we recommend to avoid `SHA-1` usage when possible.

OpenSSL. We have contacted the OpenSSL developers on December 14th. They are considering disabling `SHA-1` at security level 1 (defined as 80-bit security) after our attack. Since security level 1 is the default configuration, this would prevent `SHA-1` usage for certificates, and for handshake signatures.

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 [BL16], but this is not practical given the timing constraints.

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[14], and 9% of servers use `HMAC-SHA-1` in the record protocol[15]. This mostly corresponds to servers running old versions of SSH daemons.

7.5 Other Usages of `SHA-1`

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.

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

[12] 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

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

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

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

The GIT developers have been working on replacing SHA-1 for a while[16], and they use a collision detection library [SS17] 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 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 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 exhibited a practical chosen-prefix collision attack on `SHA-1`, and performed an actual CP collision computation for a reasonable cost. This cost will decrease over time and in a close future will be so cheap that any ill-intentioned person could afford it.

Our work directly impacts the security of PGP/GnuPG Web of Trust: using our CP collision attack on `SHA-1`, we have created two PGP keys with different UserIDs and colliding certificates. This allows an attacker to impersonate any user, as long as trusted third parties sign identity certifications with `SHA-1`.

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 100k US$ to rent GPUs and break cryptography with a security level of 64 bits (i.e. to compute 2^{64} operations of symmetric cryptography).

Future works. The cost of our attack is roughly four times the cost of a plain collision attack on the GPU we considered, so there is limited room for improvements in terms of complexity. The factor 4 could be slightly reduced by changing the parameters of the graph, but this is unlikely to result in an improvement of more than a factor 2. Alternatively, improvements to the near-collision search could improve simultaneously identical-prefix and chosen-prefix collision attacks.

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 [SBK+17], but using other interesting ones (for example the ones already used in other previous `SHA-1` attacks), we would increase the pool of available differences and in turn reduce the required number of blocks.

A natural future work would be to study other applications of practical chosen-prefix collisions on `SHA-1`. Different protocols would lead to different constraints for the attacker, and unfortunately `SHA-1` remains used on field. What would then be the security impact of our findings on these applications ? What applications could be targeted ?

Finally, it remains to be studied how recent chosen-prefix collision attacks could potentially apply to other hash functions, such as `RIPEMD`, (reduced-round) `SHA-2`, or even others.

We are planning to publish code to make our claims easy to verify, but due to the strong impact of this attack, we will wait until counter-measures are more widely implemented.

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

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

[BC04] Eli Biham and Rafi Chen. Near-collisions of SHA-0. In Matthew Franklin, editor, *CRYPTO 2004*, volume 3152 of *LNCS*, pages 290–305. Springer, Heidelberg, August 2004.

[BCJ+05] 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.

[BL16] Karthikeyan Bhargavan and Gaëtan Leurent. Transcript collision attacks: Breaking authentication in TLS, IKE and SSH. In *NDSS 2016*. The Internet Society, February 2016.

[Bra90] Gilles Brassard, editor. *CRYPTO*, volume 435 of *Lecture Notes in Computer Science*. Springer, 1990.

[CDF+07] J. Callas, L. Donnerhacke, H. Finney, D. Shaw, and R. Thayer. *RFC 4880 - OpenPGP Message Format*. Internet Activities Board, November 2007.

[Dam89] Ivan Damgård. A Design Principle for Hash Functions. In Brassard [Bra90], pages 416–427.

[DR06] 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.

[Han19] Robert J. Hansen. SKS keyserver network under attack. https://gist.github.com/rjhansen/67ab921ffb4084c865b3618d6955275f, June 2019.

[Jou04] Antoine Joux. Multicollisions in iterated hash functions. Application to cascaded constructions. In Matthew Franklin, editor, *CRYPTO 2004*, volume 3152 of *LNCS*, pages 306–316. Springer, Heidelberg, August 2004.

[JP07] 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.

[Kli06] Vlastimil Klima. Tunnels in hash functions: MD5 collisions within a minute. Cryptology ePrint Archive, Report 2006/105, 2006. http://eprint.iacr.org/2006/105.

[LP19] 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.

[Mer89] Ralph C. Merkle. One Way Hash Functions and DES. In Brassard [Bra90], pages 428–446.

[MP08] Stéphane Manuel and Thomas Peyrin. Collisions on SHA-0 in one hour. In Kaisa Nyberg, editor, *FSE 2008*, volume 5086 of *LNCS*, pages 16–35. Springer, Heidelberg, February 2008.

[Nat95] National Institute of Standards and Technology. FIPS 180-1: Secure Hash Standard, April 1995.

[Nat02] National Institute of Standards and Technology. FIPS 180-2: Secure Hash Standard, August 2002.

[Nat15] National Institute of Standards and Technology. FIPS 202: SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions, August 2015.

[Riv91] 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.

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

[SBK+17] 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.

[SKP16] 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.

[SLdW07] 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.

[SS17] 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.

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

[Ste13a] 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.

[Ste13b] 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.

[WYY05] 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.

A Example of PGP Keys for Impersonation Attack

```
-----BEGIN PGP PUBLIC KEY BLOCK-----

mQQNBH/oF4ABIAD/S2V5IGlzIHBhcnQgb2YgYSBjb2xsaXNpb24hIElOJ3MgYSB0
cmFwIXnGGvCvzAVFFdknTnMHYksdx/sjmIu43otXXbp7nqsxwWdLbZdDeKgncy/1
hRx2ouYHcrWkfOHqxAu5k8EtjHDiSk+NX83twbMsnPGeMa8kKXWdQuTf2zFxn1h2
I+5VKTm23NxFn8pTVTtw+H7eMKJH6jr2x1mi8gsyDXYNtk/0eQhP08yzzdSDYtlq
nEMGF8r/bDbGN+U/3ihBf2Jv7FTteUOkbl9XMPK7OPsd9uAJABDQDiSteL+SZBmT
YI6NFYp4nzTEb+HmAn81pMv7gnB2xQ7KDot8ymm7LCt5Aln5v5Vw3Y1EN6MRX6/3
w8rAmtJSZgVcJxBHVReOrv+CWiyqKs+13mTOdkHcWaVBqfycdWdW4uI9xxPIwkyX
kKprDjin9V8URSocooUN3ZVi/ZoYrUJJaqlwCPdGcvaO9GHriLCZM9YmtPkYdJzA
J/3dbEJfxCFoNdATTRUoW6sst4Sk98u0+1FNS/D2I3zwCp6fEyuaBm5v0X9sQph0
eFhv9lGvlnR/tCa5hyuaiOQGP1m7M0zABlD4OoDEJ1G3GXTTAPwoGaLo8eMsG1HL
GOa/xNubrvZ11Kr1sVdKBH+PbdLsFTqTQSKTl02Sj4jO2TY8/vl84udCvzTJa47z
h1Z2/qXMqOX33qC6skE9TeAO5x7gHxYr22Oer9kl5q66rmo1TvF88gWkBPvbEvxF
TUH92VzyRZZkoq0DLR2mCnMmQHXX8eDWwUA656DYYd8/5XBxiN1eB9FYm5+LZjBV
P4/DUrPgwn2oC926TGQCDQYH9nQNLoASAgwRjxlZlJl+xnAClehPhvimlPrsylEs
iZXJZuZjfmFlR1VUbBDRXshFngA85nlvVon7fAnZTYNiyM5PM5ZBHTH/9luEWef+
vMgBV5t3S1TZnD6dQpgStXi3kNZ77juY93ADOo64i5vmC8TONoSzg7AO+6KzQ2lR
+zB0/AKu92MeHNcAEQEAAdEAAAEZwFcBEAABAQAAAAAAAAAAAAAAAAAP/Y/9sAQwD/
////////////////////////////////////////////////////////////////
/////////////////////8AACwgAQABYAQERAP/EACgAAQEBAAAAAAAAAAAAAAAA
AAAEAxABAAAAAAAAAAAAAAAAAAAAAP/aAAgBAQAAPwDQTqABOoAE6gE6gATgAKAT
ihOChOKE4AAAKE4AChOKE6gATqE6gE4oTigH/9lduLZvOS5WXbOQhggKNiGWIm/z
IV/tc5s7KOrrL9gZCVAvnv1D5pEAuEqDOBDdABEBAAGOGUFsaWNlIDxhbGljZUBl
eGFtcGxlLmNvbT6JBFQEEwEIAD4WIQRDzVxbBP9XQvoUGryp11WpY1SMeAUCf+gX
gAIbLwUJAAFRgAULCQgHAgYVCgkICwIEFgIDAQIeAQIXgAAKCRCp11WpY1SMeJHD
H/9jJj4FARmtNhFXqDhNcNV6C3mQAevQR1iCbFwWHXdd2yRfyFv9/dVY4Wa2Q16j
BuCUhl7SDmO1IYlYBBo+91Kr11rDezL/y/POQsKzRp9nEoOr+QCVnIozjJ1DeP77
Xwxu77gRe1QkIql0aVUhP6vPRD+oFgViYUUWIdNgh4bBJGDMY43BJyeCxmuqRo8/
sHTWu7rmhuHpsQiTRd5valBPHXbZKAcS+TmgGM8dMv3NOldq1wDZWDrWfD6vCmnP
a4hH3UybM9G6kkkQ2xETt7hyW6XlsLXk9CtaZ2hz1kUUKtTa8LU1mj9T8dsLbTj0
qipYps/ryOCro7gkaYiNGeeelNzPhfz8jXHrK47hZ0iZbcb7JUcR6WodGAWV6bFE
+Jj02NOnX87QsWk1z5YdHsvDEbzbE9fQZ2r8hPEDsqEDXsSSw6h/FkhCcxwn9RV2
SGt5kZkU0qGP6chAITiGEEX40TcnfgyNvSJcHJ2iMaKGEaaQv5MfaefMRX9tGiz3
wmTYxSdUqkFiG2YtOkZWdCZDBKVL6Y8g7fM6G5DQc+/4alVqGdZ11TwAmKcZ142E
2JL6MAu9zfmjMcRkXytzUfkKGKa/kpAMPsn1cWYxlQe0PxxVAM8iZTPUSnBah13t
Ec1fx9jJ1cJkHBW7w1AFS+kLKD0blX9rreTUDzBQLPkaXWqH5fLZvc+EGJpt6roL
obVZ2mVDfmYNABYnw4nTDBGb2Vk60MpPh0k7pMMbXJA9kuwvgYa6VztyIZcyi8Mz
L0Kbtwsog0S5KV01tOcKMGtXpni6Vsx/k1ydvtOBUuG7VjG8Y1EuGyOUzdIuXe1r
UUTIV7GZfNpzxAumvoY2umKJQzpA/jG1yban05IViIIFotKCfA8cpKzY1roUtyjq
FPgBOT2YOadFStarrypb+f3Snj0nK4UqyJ0YbJQiKbo/NVvY2cQdqR5wyBVKV7RR
qjkNArxkHu0RUaBnywzyzwtOHNs0ojX3IyOeytLcIrbaKYJYaeZCwOWcNIWP5avq
u4Lkfq+ariSGdbfHKBR1OJWBeD/Rne+E9rmwrvk2S8fd1pdx44R8RzIvWfCKUQeu
wnfHBsP5BOZkjltSQnpBHsNSFnieOo5uSKnL8dyTbH3TtX1u3D7S95KZRley+n+5
hfQTTHWo8Xq/m6wr0TYxdA4DbaSJJ/wZFH0YaybXXkXhIQY3n3o+4mdgBQy3okIG
enWdH6Jdty6xrDimgxMjOUIyQ0JTlHBksIIViqzaY52ATI2U5UBpK7DgNC+IzIFe
QVWNIFVqT3Ifqcfd5iKZZvEEKt4QWDfIvc5bPkXraWV8Aqq3yl0iYiBFRVLVDBy3
AXrthG5XMjPFw/7TQEoIY5MYiQEcBBABAgAGBQJ/6VrwAAoJEK+7H+1pUalWFn0H
/AmChC2A4mXbw6m6hIs/zWUTNRAM4I9daUst138jmNFKUz67zIxUvUeUuU2n6Lfj
rerpVX8LuQbz2/e+g1AIcAKBRniBDyjzedWuya/faLW9rSpATUy/3Rv2G0T1QBSw
fsQNlNT8oBbMYZl7ht0ylst2m9zJyI+XjktMiIIxfr8zlMjIpXU9xF0YPuSNM0Sm
l38jeLHN9hHZpvsvsK2EmhCTF2KhHvozlFfJ8wr10cnWxdEzoYntx178Ty0KRgC+
oTLIccOSrNKjzkRuNnOsEEpAudFR6yxx8HLZatSXWpzitNdaG8APDwJiS5AHaqVQ
yNnDiF8Aoz5caQEw4D8aDvk=
=04wQ
-----END PGP PUBLIC KEY BLOCK-----
```

Figure 9: Key A

```
-----BEGIN PGP PUBLIC KEY BLOCK-----

mQMNBH/oF4ABGAD/UHJhY3RpY2FsIFNIQS0xIGNob3Nlbi1wcmVmaXggY29sbGlz
aW9uIR0nbGumYeEEDh99dn8HYkndx/szLIu4wrdXXb7Hnqsr4WdLfbNDeLTLcy/h
iRx2oCYHcqUQfOH26Au5l30tjGhSSk+dX83tzQssnOGSMa8m6XWdUlDf2y1Nn1hy
n+5VMxm23Mxhn8pPuTtw7HLeMKCH6jrmc1mi7icyDXKxtk/syQhPw8yzzdg7Ytl6
kEMGFQr/bCZyN+Uj4ihBe95v7E7NeUO0Sl9XLB67OO8R9uALwBDQHpCteKO+ZBmX
3I6NDTp4nyTEb+Hqun81tMf7gnK2xQ7auot81lW7LC/FAlnjn5VwzalEN7/9X6/j
z8rAmBJSZhXoJxBbeReOqkOCWjQaKs+l3mTOevncWbVNqfyetWdW8lY9xw/0wkyT
LKprFBin9U8wRSoAToUNyZli/ZjYrUJZ3qlwFNtGcvIy9GHzOLCZI9YmtPWgdJzQ
K/3dboJfxDHcNdAPcRUoXxcst56E98uk31FNVxz2I2j8Cp6d0yuaFtpv0WNAQphw
xFhv7uGvlmR/tCa1PyuamOgGP1t7M0zQslD4JrzEJ1ULGXTJIPwoCYbo8f/AG1Hf
FOa/xhubrubB1KrpnVdKAMOPbcpcFTqDQSKTm/WSj5jC2TY+Pvl88lNCvyj1a473
O1Z25IXMqPXT3qCmXkE9WewO5xwgHxY7b2Oes/Ul5qoGrmot/vF84gWkBPdjEvxV
QUH925zyRYbQoq0fER2mDs8mQG/38eDG5UA6+0zYYcsz5XBzSN1eF2VYm4OnZjBR
g4/DSgPgwm2oC9229GQCHQYH9nQNLoASAgwRjxlZlJl+xnAClehPhvimlPrsylEs
iZXJZuZjfmFlR1VUbBDRXshFngA85nlvVon7fAnZTYNiyM5PM5ZBHTH/9luEWef+
vMgBV5t3S1TZnD6dQpgStXi3kNZ77juY93ADOo64i5vmC8TONoSzg7AO+6KzQ2lR
+zB0/AKu92MeHNcAEQEAAbQVQm9iIDxib2JAZXhhbXBsZS5jb20+iQNUBBMBCAA+
FiEExr/i/LvlGokr63eYEjPUzGHb2cQFAn/oF4ACGy8FCQABUYAFCwkIBwIGFQoJ
CAsCBBYCAwECHgECF4AACgkQEjPUzGHb2cSLZhgAynF426xC9qX3Afh2jzWUFWMZ
duiZ78JwpopGRo/BnAzlD/dmmK/OjaVfG7aCq/R/Fd3faeA1qxJuST6Z2bwDCbtj
HjERNPATekCJaHP4EgHsFSDZQUNdUWswO7rclXnHmN7AxCkfXg+U5R1cxQkyBIL7
U9uxleMU6+Fr+QP2y4MsZpM1n1ipWJK1FuKKQeNY9J7fRdr0LyqiVfYOTqtI6kr2
tzk1oQ2CsiRmidC0MmAaW1L5tppmDiPMo3Yqv0lCrqeyfnOBpwOLFIz9DH6Ndf1z
213eDWgUpHbHvERS0rwxv/4eizUGmisxgjrWwjhdRch7ZNWQZYQVwtcjz7wWhVPW
vaS+t86YayrDO8J1jw7FfkFdPHlt1TlawWcGsiehiB9EazJFM4zGw/IYjQWLhBvv
e+NeKrSjIso3ZXSbqVdl/Dgj4Go9J3KXXqJSbaxedak3SX44XwO54NAgYsjl0ak5
yHKdBjrZYdtElzkkJklUEYdQyA5icRjJ2Kiy3g4xlR+7ev6i1QSoouBIbQD7qmeO
45iKG17m0e6qMPrPB9sTUue/gVfZaysR4nUX1zIIk6sHg2Q7Fl1ohYO5SFYvFVrj
dtpGO1XeKXnEKaUb9Z90CUlh16SRL7/xXq86SKqi7SoIMNFspUmsuk4np/e6ipBn
u2cAjpThNrELZjmvh0KCt/x6jv1vbdUiB+gg2SbTD5qFmoir2OXfKlNrz6PUhGI4
dQFKbBsmbJtUqTDEogRlHUZ2CXWpj5Zs9xvcGvaTpurqncoEmz96SKyhiPSpDw0x
tIxEaupvGnug4/CtlvFE+vP7ZzucMSThLLXJVRATNWcCWosfXXWv4A7Lebbxqj2P
ukwJFmmJOet8s2h6blFRm3MegHP7hqjqQe5XovZLqwLKrr3DqdZYhmx0jeQAToII
BKvviJnh8XU293hHv8M4wGRVPvyMrTfwcMj+pgb4yTPShoysONWIE954c3yic94+
slaK45askyHC9ORw8oraPOyEOI9ewSfi5N3zwXImiQEcBBABAgAGBQJ/6VrwAAoJ
EK+7H+1pUalWI6UH/3HgRCkMiyDRrRi8lZrMWcf0xIlC1kiSb62Kt2VhsG/RL2tO
GJZhrOp3+fg3QiRFMnE+5ScJIBhL3PgScTEOoCcmA3+jo7s8fnfhLVzbHjcJEtxP
fo4zlr9pKh5cUXlsSrZPXCHE/1IdEXjbETie6fS9aSWhnDYnBdwTw8CrHRY/tibn
tMla3m8w89zfTcDVkHNSgj6nSQly+jlU+3I5Ny+pJGxEytMeBwyWoC0nR/qltm7U
kG2yMV83tqkSVqMyIPGx5MQuc21sDWTmP3ct+VMrJD448+S2Emor0InhmkRuT5bS
7p8AHuEkhlPITSRvGpgQjnzDGuP6TUbVUGReTMvRwFnAVwEQAAEBAAAAAAAAAAAA
AAAA/9j/2wBDAP//////////////////////////////////////////////////
////////////////////////////////////wAALCABAAFgBAREA/8QAKAABAQEA
AAAAAAAAAAAAAAAAAAQDEAEAAAAAAAAAAAAAAAAAAAAA/9oACAEBAAA/ANBOoAE6
gATqATqABOAAoBOKE4KE4oTgAAAoTgAKE4oTqABOoTqATihOKAf/2V24tm85LlZd
vRCGCAo2IZYib/MhX+1zmzso6usv2BkJUC+e/UPmkQC4SoM4EN0AEQEAAbQAAAAZ
QWxpY2UgPGFsaWNlQGV4YW1wbGUuY29tPokDVAQTAQgAPhYhBMa/4vy75RqJK+t3
mBIz1Mxh29nEBQJ/6BeAAhsvBQkAAVGABQsJCAcCBhUKCQgLAgQWAgMBAh4BAheA
AAoJEBIz1Mxh29nEsnkYALms73X14/+0sUPjojsWYeUzFSY67/ihObZBEMcp3qoz
4E5kFlg2ZquAEBnMa3IANy7kvZprxgYd3EABzaBUFkMsrGov0aaJCLmtSvWP9ooh
0vWgHIOtuTUZGBcdJhNORnZ/BOW1azlrYGdzIiQOzMn+67BfUkJIIoHgdNOOZZkE
f2fncC3vmyh9/8QhsUF553mOxo8VYYNS0/S/GSr6MD1eP+YpA3v2DznaHyTzwFmN
l4apcEErudob/PSzn9SdHDjwC2WxqObEWllLVtnANi1GP+sSTVbWCREJ+m72VljV
ADP5nKD3HPKeqoRNuQkv2TdQdkyTp3UBechnqK+u7xl72d0ZGJ9EOnT/o23zbxFz
vrIcBbCQnqrtZnwUDIaxG8numABkfM6T5m0z0etIOctQoRw7MQsB4C6VxqDtho3k
5dSomtsiNsiuLlI6FT5YIUJWOV7r8UVzrAKq98buaAHlnBhpps2WHBrm8o0BO16g
rJOEfhgYyB6cMG60hLAhkjPVbu5HxBngylu1ZrotktK+eWGtWyZRA6/pZN5t7EdO
IzOiS6Ewp5yBHB5t24ZytrZhVPgExOka47yEJMKq8Q+uM8V3kfsk6+Kb6aK2cJyg
A1igSL9T48a9es0dWm+rPxKg2Sb1+QdWCWpG52Nb4d+EfqQERdIgp97dgJaZ3ye7
ORR+tK9gpoJHDI1UBdm3M8PeYhiMbOWWvuIsuB5+/fstLTNngmyT+gqwr4F3XmG/
3ZoclgmKxRG7VYn3DiT7PLZJmSovpuGay2pqAQvc+nJrjC0w67qUmBm8QHwbMWMh
xZkfBLfD6V/7iWCIhPDmJteKW/pxBASJh9gA9h8T3BRTTOSM252aaG6EYF5Wc+ha
/rhl1c6xSE6SEArf+aYcrz4KXap4HDC4pYqJIltWg8uvs2+qejYH4iW3KArmmfjU
3naa7LDUKEs7U2tcPyg1ZfTC529sROq50gJ1agcaHr8RJdIZ6qGVdaBQLMG9utUW
60uHZmhCPy+lS3Ftjp19/YkBHAQQAQIABgUCf+la8AAKCRCvux/taVGpVhZ9B/wJ
goQtgOJl28OpuoSLP81lEzUQDOCPXWlLLdd/I5jRSlM+u8yMVL1HlLlNp+i3463q
6VV/C7kG89v3voNQCHACgUZ4gQ8o83nVrsmv32i1va0qQE1Mv90b9htE9UAUsH7E
DZTU/KAWzGGZe4bdMpbLdpvcyciPl45LTIiCMX6/M5TIyKV1PcRdGD7kjTNEppd/
I3ixzfYR2ab7L7CthJoQkxdioR76M5RXyfMK9dHJ1sXRM6GJ7cde/E8tCkYAvqEy
yHHDkqzSo85EbjZzrBBKQLnRUesscfBy2WrUl1qc4rTXWhvADw8CYkuQB2qlUMjZ
w4hfAKM+XGkBMOA/Gg75
=T/Dd
-----END PGP PUBLIC KEY BLOCK-----
```

Figure 10: Key B