A Survey of Chosen-Prefix Collision Attacks

Marc Stevens

Cryptology Group, CWI, Amsterdam, The Netherlands ⋆

1 Cryptographic hash functions

A cryptographic hash function $H : \{0,1\}^* \to \{0,1\}^n$ is a function that computes a fixed-length hash value of n bits for any arbitrary long input message M. For cryptographic purposes, H should satisfy at least the following security properties:

- *pre-image resistance*: given a hash value h, it should be infeasible to find a message x such that $H(x) = h$;
- *second pre-image resistance*: given a message x, it should be infeasible to find a message $y \neq x$ such that $H(x) = H(y)$
- *collision resistance*: it should be infeasible to find collisions, i.e., two distinct messages x and y that hash to the same value $H(x) = H(y)$.

Cryptographic hash functions are the swiss army knives within cryptography. They are used in many applications including digital signature schemes, message authentication codes, password hashing, cryptocurrencies and content-addressable storage. The security or even the proper functioning of these applications relies on the security property that is the main focus of this chapter: collision resistance. For instance, all major digital signature schemes rely on the hash-then-sign paradigm. This implies that for any colliding pair $x \neq y$ with $H(x) = H(y)$, any signature for x is also an unwanted valid signature for y, and vice versa. When finding meaningful collision pairs (x, y) is practical, this can have grave implications as will become clear below.

Collision resistance. The best generic attack to find collisions for a cryptographic hash function is the birthday search. For an output hash of n bits, on average $\sqrt{\pi/2} \cdot 2^{n/2}$ hash evaluations are needed to find a collision among them. This can be achieved in practice using little memory and is easily parallelisable [vW99]. We call a cryptographic hash function collision resistant if it is not possible to find collisions faster than this generic attack. In contrast to essentially all security properties of other cryptographic functionalities, collision resistance of a cryptographic hash function defies formalization as a mathetical security property. Such formalizations dictate the non-existence of an efficient algorithm that solves the security problem with non-negligbile probability. The underlying issue, dubbed the foundations of hashing dilemma [Rog06], is the following. For any given cryptographic hash function H there exists a collision $x \neq y$ with $H(x) = H(y)$ by the pigeon hole principle. This implies there actually exists a trivial

⋆ This material will be published in revised form in Computational Cryptography edited by Joppe W. Bos and Martijn Stam and published by Cambridge University Press. See www.cambridge.org/9781108795937.

collision finding algorithm $A_{x,y}$ that succeeds with probability 1 by simply printing a collision pair x, y. Collision resistance thus remains an informal security property that there doesn't exist a *known* algorithm to find collisions for H faster than the generic attack.

Standards. In practice, the SHA-2 and SHA-3 families of hash functions, each consisting of 4 hash functions with output sizes 224, 256, 384 and 512 bits, are current NIST standards that are recommended for cryptographic purposes. Their predecessors, namely the MD5 and SHA-1 hash functions, have been used for decades as the software industry's de facto standards.

With the exception of the recent SHA-3 family, MD5, SHA-1 and SHA-2 are all based on the Merkle-Damgård framework [Dam90, Mer90]. This framework builds a cryptographic hash function H for arbitrary-size inputs from a compression function f with fixed-size inputs

$$f : \{0,1\}^n \times \{0,1\}^m \to \{0,1\}^n.$$

The hash function initializes a chaining value CV_0 with a fixed public value IV and iteratively updates this chaining value using f:

$$CV_0 = IV; \quad CV_i = f(CV_{i-1}, m_i), \quad \text{for } i = 1, \ldots, k$$

The final chaining value CV_k is used as the hash output $H(M) := CV_k$ of M.

The Merkle-Damgård construction admits a security reduction of the collision resistance of H to the collision resistance of f, i.e., it implies that it is at least as hard to find collisions for the hash function H as it is hard to find collisions for its compression function f. Hence, an early practical collision attack on MD5's compression function by den Boer and Bosselaers [dB94], should have been an important warning against MD5's widespread adoption till 2004...

Collision attacks. The first collision attack for MD5 was presented in 2004 by Wang *et al.* [WY05], together with an actual example collision consisting of two different 128-byte random looking files. Their collision attack can be used to craft so-called *identical-prefix collisions (IPC)* of the form $H(P\|C_1\|S) = H(P\|C_2\|S)$, where P is a shared prefix with byte length a multiple of 64, $C_1 \neq C_2$ two 128-byte collision bit strings (dependent of P), and S a shared suffix.

As identical-prefix collisions have differences only in 128 byte random-looking strings C_1 and C_2, these initially didn't seem very harmful for real world applications. Yet several abuse scenarios for identical-prefix collisions were quickly demonstrated, e.g., to mislead integrity software [Mik04, Kam04] in 2004, X.509 certificates with distinct public keys [LdW05] in 2005, visually distinct Word, TIFF, and black&white PDF files [GIS06] in 2006. Though these were working proof of concepts of meaningful collisions, due to their limitations, they did not form very convincing real world threats.

In 2007 a stronger and more expensive attack called *chosen-prefix collision (CPC) attack* [SLdW07c] was introduced that can produce collisions of the form $H(P_1\|C_1\|S) = H(P_2\|C_2\|S)$, where the prefixes P_1 and P_2 can be arbitrarily and independently chosen. The main topic of this chapter is the development of practical chosen-prefix collision attacks and their applications. A historical overview of the identical-prefix and chosen-prefix collision attack costs for MD5 and SHA-1 can be found in Tables 1 and 2.

Year	Identical-prefix collision cost	Chosen-prefix collision cost
< 2004	2^{64} generic	2^{64} generic
2004	**2^{40} [WY05]**	–
2005	2^{37} [Kli05]	–
2006	2^{32} [Kli06, Ste06]	**2^{49} [SLdW07c]**
2007	2^{25} [Ste07]	–
2008	2^{21} [XLF08]	–
2009	2^{16} [SSA+09]	2^{39} [SSA+09]
2020	2^{16} [SSA+09]	2^{39} [SSA+09]

Table 1: Historical overview of MD5 collision attack complexities. (bold=first collision example)

Year	Identical-prefix collision cost	Chosen-prefix collision cost
< 2005	2^{80} generic	2^{80} generic
2005	2^{69} [WYY05b]	–
	(u : 2^{63} [WYY05a])	–
2007	(u : 2^{61} [MRR07])	–
2009	(w : 2^{52} [MHP09])	–
2013	2^{61} [Ste13b]	2^{77} [Ste13b]
2017	**G : $2^{63.1}$ [SBK+17]**	–
2019	–	G : 2^{67} [LP19]
2020	2^{61} / G : $2^{61.2}$ [Ste13b] / [LP20]	**G : $2^{63.4}$ [LP20]**

Table 2: Historical overview of SHA-1 collision attack complexities. (u=unpublished, w=withdrawn, G=using GPU, bold=first collision example)

Near-collision attacks. Both identical-prefix collisions and chosen-prefix collisions are built from *near-collision attacks* on the compression function. Differential cryptanalysis is used to construct such attacks against the compression function, where differences between two related evaluations $f(CV, B)$ and $f(CV', B')$ are analyzed. For any variable X in the evaluation of $f(CV, B)$, we denote by X' the respective variable in the evaluation of $f(CV', B')$ and by $\delta X = X' - X$ the difference between these two variables.

The compression function is essentially a block cipher $E_K(M)$ in Davies-Meyer feed-forward mode $f(CV, B) = CV + E_B(CV)$. More specifically, a near-collision attack employs a *differential characteristic* against the block cipher, i.e. a valid trail of differences for all internal values of all rounds of the compression function. The first part of the differential characteristic over about $1/4$-th of the rounds can be dense and is required to start with the given input chaining value difference. But it is the last $3/4$-th of the rounds that is the critical part that mostly determines the attack cost, which needs to be optimized: sparse and high probability. Due to Davies-Meyer, the final difference D of the differential characteristic is added to the input chaining value difference: $\delta CV_1 = \delta CV_0 + D$.

For instance, an identical-prefix collision requires a partial differential characteristic over the last $3/4$-ths of the rounds with high probability. This partial characteristic with

final difference D will be used for both near-collision attacks as follows. The first near-collision attack begins with chaining value difference $\delta CV_0 = 0$. An appropriate full differential characteristic is used, which can be precomputed, and dedicated collision search techniques are used to find a solution: a pair of blocks that fulfills the differential characteristic. After this first attack a chaining value difference $\delta CV_1 = \delta CV_0 + D = D$ is obtained.

The second attack continues with $\delta CV_1 = D$ and now instead 'subtracts' the difference D. It reuses the exact same critical part of the differential characteristic of the first block except all differences are negated. The first part of the second attack's differential characteristic needs to constructed for the input chaining value difference $\delta CV_1 = D$. After the second near-collision attack an internal-state collision $\delta CV_2 = \delta CV_1 - D = 0$ is achieved. Due to the Merkle-Damgård structure any internal-state collision directly results in a hash collision, even after appending an arbitrary shared suffix S.

2 Chosen-prefix collisions

Chosen-prefix collision attacks were introduced by Stevens, Lenstra and de Weger [SLdW07c] in 2007. Given two distinct prefixes P and P' of equal bit-length $|P| = |P'|$ a multiple of the blocksize of 512 bits, a chosen-prefix collision attack constructs two suffixes $C \neq C'$ such that $H(P\|C) = H(P'\|C')$ in general in two phases:

1. *Birthday search phase* to find bitstrings B and B' such that appending these to P and P' results in a chaining value difference in a target set $\mathcal{D}$:

$$\delta CV_0 = \hat{H}(P\|B) - \hat{H}(P'\|B') \in \mathcal{D},$$

   where $\mathcal{D}$ is determined by the next phase.[1]
2. *Near-collision attacks phase* to find a sequence of near-collision attack block pairs $(M_0, M'_0), \ldots, (M_k, M'_k)$ that each add specific differences to the latest chaining value difference in order to reduce the chaining value difference δCV_0 to 0 in the end:

$$\hat{H}(P\|B\|M_0\|\cdots\|M_k) = \hat{H}(P'\|B'\|M'_0\|\cdots\|M'_k).$$

The desired suffixes are thus $C = B\|M_0\|\cdots\|M_k$ and $C' = B'\|M'_0\|\cdots\|M'_k$. These two phases are elaborated below.

Birthday search phase. Van Oorschot and Wiener [vW99] remains the state-of-the-art on how a birthday search can be executed in practice. This is done by iterating a properly chosen deterministic function $f : V \to V$ on a certain space $V = \{0,1\}^k$, assuming that the trail of points of V thus visited form a pseudo-random walk. To take advantage of parallelism, many different pseudo-random walks are generated, of which only the startpoints, lengths and endpoints are kept. All walks end at a 'distinguished point', these form a subset of points with an easily recognizable pattern that is chosen

[1] By $\hat{H}$ we mean the hash function H without any padding for all messages of bit length a multiple of 512 bits, thus resulting in the chaining value directly after processing the entire input.

depending on $|V|$, available storage and other characteristics. Since any two walks that intersect due to a collision will have the same endpoint, they can be easily detected. The collision point can then be recomputed given the startpoints and lengths of the two colliding trails.

If there are additional constraints that could not be encoded into the choice of f and V then the birthday search has to continue until a collision point has been found that satisfies these additional constraints. Assume a collision point has a probability of p of satisfying these additional constraints, then on average one has to find $1/p$ collision points. In this case the expected total cost C_{total} is:

$$C_{total} = C_{tr} + C_{coll}, \quad C_{tr} = \sqrt{\frac{\pi \cdot |V|}{2 \cdot p}}, \quad C_{coll} = \frac{2.5 \cdot C_{tr}}{p \cdot M},$$

where C_{tr} is the cost before a collision occurs and C_{coll} is the cost of recomputing the collision point(s) and M is the average amount of stored walks. It follows that to ensure $C_{coll} \ll C_{tr}$, i.e., the overhead in recomputing costs remains a small fraction of the total cost, one needs to ensure $M \gg 2.5/p$ by choosing a suitable distinuished points pattern.

In the case of a chosen-prefix collision attack, given the set $\mathcal{D}$ of target chaining value differences we can expect a complexity $C_{tr} \approx \sqrt{\pi \cdot 2^n/|\mathcal{D}|}$ by appropriately choosing f and V as follows. We need to partition V into two equally sized disjoint subsets V_1 and V_2 in order to map half of the inputs to a computation related to prefix P and the other half to P', e.g., by partitioning on the first bit of each $v \in V = \{0,1\}^k$. The function f is thus actually split into two functions $f_1 : V_1 \to V$ and $f_2 : V_2 \to V$, and each typically is composed of three functions: $f_i = g_i \circ \mathit{MD5Compress} \circ h_i$:

- $h_i : V \to \{0,1\}^n \times \{0,1\}^{512}$ maps every point to a compression function input pair (CV, B) of chaining value CV and message block B. E.g., $h_i(v) = (CV_i, 0^{512-k} \| v)$, where $CV_1 = \hat{H}(P)$ and $CV_2 = \hat{H}(P')$ (see [1]). [2]
- $g_i : \{0,1\}^n \to V$ maps a chaining value x to an element of V by taking a fixed subsequence of k bits of x indexed by I denoted as $v = (x_j)_{j \in I}$, which is the same for both g_1 and g_2. To allow more efficient encodings, one can consider g_2 first adding a fixed offset d to x: $v = ((x + d)_j)_{j \in I}$. Given bit-selection I and offset d one can efficiently approximate the corresponding probability $p_{I,d}$ of a collision point being useful: [3]

  $$p_{I,d} := 1/2 \cdot \Pr[y - x \in \mathcal{D} \mid (y_j)_{j \in I} = ((x + d)_j)_{j \in I}]$$

  The goal is to choose (I, d) with high probability $p_{I,d}$, ensuring low memory requirements, from those that achieve near-optimal birthday search cost:

  $$\sqrt{\pi \cdot 2^{|I|}/(2 \cdot p_{I,d})} \approx \sqrt{\pi \cdot 2^n/|\mathcal{D}|}.$$

[2] Instead of prepending zero bits, one can also use any (more complex) injective map onto the block space $\{0,1\}^{512}$. It is even possible to generalize to encode the input x into the entire prefix P_x or P'_x, but this is not advisable as this will require many *MD5Compress* calls per f-call and significantly increase the birthday search cost.

[3] Besides requiring $f(y) - f(x) \in \mathcal{D}$, we additionally need these inputs belonging to different prefixes, i.e. $x \in V_1 \Leftrightarrow y \in V_2$, which happens with probability $p = 1/2$.

Example I and d as chosen for actual chosen-prefix collisions attacks are shown in Section 4.

Near-collision attacks phase. Once the above birthday search has succeeded in finding messages $P\|B$ and $P'\|B'$ with a target chaining value difference $\delta CV_0 = \hat{H}(P'\|B') - \hat{H}(P\|B) \in \mathcal{D}$, the next goal is find a sequence of near-collision blocks that reduce the chaining value difference to 0. These attacks are designed in the following steps:

1. *First, one needs one or more high probability core differential characteristics over all but the first* $1/4$*-th of the rounds.* The success probability of this part mainly determines the near-collision attack complexity. For MD5, the main family of core differential characteristics considered is based on message block difference $\delta m_{11} = \pm 2^b$ and $\delta m_i = 0$ for $i \neq 11$, see Table 8. However, MD5 chosen-prefix collision attacks with other core differential characteristics have been demonstrated [SSA$^+$09, FS15].
2. *Second, one determines which set of output differences to use for each core characteristic.* Each of the chosen core differential characteristics $\mathcal{P}$ results in a single output difference $D_{\mathcal{P}}$ (on top of which the Davies-Meyer feedforward adds the input chaining value difference). To significantly increase the set $\mathcal{D}$ beyond these $D_{\mathcal{P}}$, one chooses a lower bound on the differential characteristic probability p_{min}. Next, one searches for variants of the core differential characteristics $\mathcal{P}'$ where only the very last few steps are changed (resulting in different output difference $D_{\mathcal{P}'}$) with success probability $p \geq p_{min}$.
3. *Third, a suitable strategy has to be chosen.* During the actual attack one has to perform a series of near-collision attacks and one needs an overall strategy how to choose for each near-collision attack which one or more variant(s) $\mathcal{P}'$ of a single core $\mathcal{P}$ to use. The most common strategy is a systematic one, where essentially each bit position of a given δCV_0 with a possible non-zero difference is assigned to a specific core $\mathcal{P}$. In this case determining the set of variant characteristics $\mathcal{P}'$ whose output differences sum up to $-\delta CV_0$ is an almost trivial matter, and one can execute these in arbitrary order (cf. [SLdW07c, SSA$^+$09, FS15]). When there exist many distinct combinations of output differences that sum up to $-\delta CV_0$, a more complex strategy may exploit this to achieve a lower overall attack complexity as described in [LP19]. Based on the resulting set of output differences $\mathcal{D}$ that may be cancelled in this manner, one can now determine suitable values for I, d for birthday search step function as described earlier.

After the birthday search that has resulted in a suitable chaining value difference δCV_0, one applies the chosen strategy and constructs and executes a sequence of near-collisions. For each near-collision attack one chooses a differential characteristic $\mathcal{P}$ based on the strategy, and executes the near-collision attack in the following steps:

1. *Construct full differential characteristic* $\mathcal{P}_{full}$ *over all steps.* The initial state is determined by the current input chaining value pair, while most later steps are fixed by $\mathcal{P}_{full}$. What remains is to find a partial differential characteristic that connects these two. The first successful connection by Wang et al. was done entirely by

hand. Since then two main algorithmic approaches have been developed: guess-and-determine proposed by De Cannière and Rechberger [DR06] and the meet-in-the-middle approach [SLdW07c, Ste12, Ste09b]. In the case of SHA-1, even if the found partial differential characteristic is locally valid over those steps, there may be global incompatibilities due to dependencies between the expanded message difference equations over all steps. This has been successfully addressed by using a SAT solver to tweak the full differential characteristic to find a complete valid differential characteristic [SBK+17]. Finally, the full differential characteristic between two compression function evaluations can be transformed into an equivalent set of conditions on a single compression function evaluation.

2. *Search for a suitable combination of collision search speed-up techniques.* The basic idea behind these techniques in collision search is to change a currently considered message block value using available remaining freedom that does not interfere with the conditions of the full differential characteristic up to some step (preferably as far as possible), possibly making use of additional conditions to make it work with high probability. The most simple is called neutral bits [BC04] which only flips a single message bit. Advanced message modification [WY05] / tunnels [Kli06] / boomerangs [JP07] aim to cover more steps by changing message and state bits together, essentially applying a local very sparse differential characteristic that is independent of $\mathcal{P}_{full}$.
3. *Search for a solution.* Finally, the actual near-collision search is most commonly implemented as a depth-first tree search. Where each level of the tree is related to a single step or a (combination of) message modifications, where for each level the degrees of freedom creates new branches that may be immediately pruned by the conditions. The first phase over the first, say 16, levels consists of finding a solution for the first steps that uses entire the input message block exactly once, which is typically quite easy and cheap. But if it is especially dense in number of conditions it may be that no solutions are possible, in which case another variant differential characteristic needs to be tried. The second phase over the next few levels consists of applying speed-up techniques that modify the current solution up to a certain step in a very controlled manner resulting in another solution up to that step. In the last phase, no degrees of freedom are left and it remains to verify all remaining conditions which are fulfilled probabilitistically, hence all conditions in this phase directly contribute to the collision search's complexity.

To achieve this, one needs a family of near-collision attacks, of which the first $1/4$-th of the rounds are adjusted to the problem. This can be a family built around a single core differential characteristic, where the final few rounds are adjusted to allow for a large set of final differences at a cost in increased attack complexity. Or even better, a family built around several core differential characteristics, each with many variant final few rounds and final differences.

3 Chosen-prefix collision abuse scenarios

When exploiting collisions in real world applications two major obstacles must be overcome.

– The problem of constructing *meaningful collisions*. Given current methods, collisions require appendages consisting of unpredictable and mostly uncontrollable bit strings. These must be hidden in the usually heavily formatted application data structure without raising suspicion.
– The problem of constructing *realistic attack scenarios*. As we do not have effective attacks against MD5's (second) pre-image resistance but only collision attacks, we cannot target existing MD5 hash values. In particular, the colliding data structures must be generated simultaneously, along with their shared hash, by the adversary.

In this section several chosen-prefix collision applications are surveyed where these problems are addressed with varying degrees of success:

– Section 3.1: attacks on X.509 certificates
– Section 3.2: rogue X.509 Certification Authority certificate
– Section 3.3: supermalware FLAME's malicious Windows Update
– Section 3.4: Breaking the PGP web of trust with colliding keys
– Section 3.5: Man-in-the-middle attacks on TLS, SSH
– Section 3.6: Nostradamus attack on hash-based commitments
– Section 3.7: Colliding documents
– Section 3.8: PDF files and a practical Nostradamus attack
– Section 3.9: Colliding executables & software integrity checking
– Section 3.10: Digital forensics
– Section 3.11: Peer to peer software
– Section 3.12: Content addressed storage
– Section 3.13: Polyglots: multi-format collisions
– Section 3.14: Hashquines: embedding the MD5 hash in documents

3.1 Digital certificates

In [LdW05] it was shown how identical-prefix collisions can be used to construct colliding X.509 certificates with different RSA moduli but identical Distinguished Names. Here the RSA moduli absorbed the random-looking near-collision blocks, thus inconspicuously and elegantly solving the meaningfulness problem. Identical Distinguished Names does not enable very realistic threat scenarios. Different Distinguished Names can be achieved using chosen-prefix collisions, as was have shown in [SLdW07c]. The certificates of these forms do not contain any spurious bits, so superficial inspection at bit level of either of the certificates does not reveal the existence of a sibling certificate that collides with it signature-wise. Below in Sections 3.2, 3.3 and 3.4 are additional interesting colliding certificate constructions that are more intricate and achieve different certificate usage properties.

3.2 Creating a rogue Certification Authority certificate

One of the most impactful demonstrations of the threat of collision attacks in the real world, was the construction of a rogue Certification Authority (CA) [SSA+09]. It thus directly undermined the core of PKI: to provide a relying party with trust, beyond reasonable cryptographic doubt, that the person identified by the Distinguished Name field

has exclusive control over the private key corresponding to the public key in the certificate. With the private key of the rogue CA, and the rogue CA certificate having a valid signature of a commercial CA that was trusted by all major browsers (at that time), 'trusted' certificates could be created at will. Any website secured using TLS can be impersonated using a rogue certificate issued by a rogue CA. This is irrespective of which CA issued the website's true certificate and of any property of that certificate (such as the hash function it is based upon – SHA-256 is not any better in this context than MD4) (although nowadays this can be thwarted using so-called *certificate pinning*). Combined with redirection attacks where `http` requests are redirected to rogue web servers, this leads to virtually undetectable phishing attacks.

In fact, any application involving a Certification Authority that provides certificates with sufficiently predictable serial number and validity period and using a non-collision-resistant hash function may be vulnerable, e.g. see Section 3.3. This type of attack relies on the ability to predict the content of the certificate fields inserted by the CA upon certification: if the prediction is correct with non-negligible probability, a rogue certificate can be generated with the same non-negligible probability. Irrespective of the weaknesses of the cryptographic hash function used for digital signature generation, our type of attack becomes effectively impossible if the CA adds a sufficient amount of fresh randomness to the certificate fields before the actual collision bit strings hidden in the public key fields.

To compare with prior X.509 certificate collisions, the first colliding X.509 certificate construction was based on an identical-prefix collision, and resulted in two certificates with different public keys, but identical Distinguished Name fields (cf. Section 3.1). Then as a first application of chosen-prefix collisions it was showed how the Distinguished Name fields could be chosen differently as well [SLdW07c]. The rogue CA goes one step further by also allowing different "basic constraints" fields, where one of the certificates is an ordinary website certificate, but the other one a CA certificate. Unlike the previous colliding certificate constructions where the CA was under the researcher' control, a commercial CA provided the digital signature for the (legitimate) website certificate.

In short, the following weaknesses of the commercial CA that carried out the legitimate certification request were exploited:

- Its usage of the cryptographic hash function MD5 to generate digital signatures for new certificates.
- Its fully automated way to process online certification requests that fails to recognize anomalous behavior of requesting parties.
- Its usage of sequential serial numbers and its usage of validity periods that are determined entirely by the date and time in seconds at which the certification request is processed.
- Its failure to enforce, by means of the "basic constraints" field in its own certificate, a limit on the length of the chain of certificates that it can sign. Which when properly set could potentially invalidate any certificate signed by the rogue-CA, if clients properly enforce this chain length constraint.

Results were disclosed to the relevant parties to ensure this vulnerability was closed prior publication. This was done privately and anonymously using the Electronic Fron-

tier Foundation as an intermediary in order to reduce the not insignificant risk of cumbursome legal procedures to stifle or delay publication. After disclosure, CAs moved quickly and adequately. MD5 was quickly phased-out from all public CAs, and later formally forbidden in the CA/Browser Forum's Baseline Requirements. Furthermore, to prevent similar attacks with other hash functions in use, such as SHA-1 (till 2019), a formal requirement was added that certificate serial numbers must be unpredictable and contain at least 64 bits of entropy. Besides the impact on Certificate Authorities, this work clearly and strongly demonstrated the significant real-world threat of chosen-prefix collisions in general and created a stronger push for the deprecation of MD5 in other applications.

Certificate construction. We summarize the construction of the colliding certificates in the sequence of steps below, and then describe each step in more detail.

1. Construction of templates for the two to-be-signed parts, as outlined in Table 3. Note that we distinguish between a 'legitimate' to-be-signed part on the left hand side, and a 'rogue' to-be-signed part on the other side.
2. Prediction of serial number and validity period for the legitimate part, thereby completing the chosen prefixes of both to-be-signed parts.
3. Computation of the two different collision-causing appendages.
4. Computation of a single collision-maintaining appendage that will be appended to both sides, thereby completing both to-be-signed parts.
5. Request certification of the legitimate to-be-signed part, if unsuccessful return to Step 2.

	legitimate website cert (A)	rogue CA cert (B)
prefix	serial number A	serial number B
	signing CA name	signing CA name
	validity period A	validity period B
	long domain name A	rogue CA name B
	"	RSA public key B
	"	X.509v3 ext: CA=true
	start RSA public key A	start X.509 comment
collision	collision data A	collision data B
common suffix	tail of RSA public key	<copy from A>
	X.509v3 ext: CA=false	<copy from A>

Table 3: The to-be-signed parts of the colliding certificates

Step 1. Templates for the to-be-signed parts. Table 3 shows the templates for the to-be-signed parts of the legitimate and rogue certificates. On the legitimate side, the chosen prefix contains space for serial number and validity period, along with the exact Distinguished Name of the commercial CA where the certification request will be

submitted. This is followed by a subject Distinguished Name that contains a legitimate website domain name (`i.broke.the.internet.and.all.i.got.was.this.t-shirt.phreedom.org`) consisting of as many characters as allowed by the commercial CA (in this case 64), and concluded by the first 208 bits of an RSA modulus, the latter all chosen at random after the leading '1'-bit.

The corresponding bits on the rogue side contain an arbitrarily chosen serial number, the same commercial CA's Distinguished Name, an arbitrarily chosen validity period (actually chosen as indicating "August 2004", to avoid abuse of the rogue certificate), a short rogue CA name, a 1024-bit RSA public key generated using standard software, and the beginning of the X.509v3 extension fields. One of these fields is the "basic constraints" field, a bit that was set to indicate that the rogue certificate will be a CA certificate (in Table 3 this bit is denoted by "CA=TRUE").

At this point the entire chosen prefix is known on the rogue side, but on the legitimate side predictions for the serial number and validity period still need to be decided which is done in Step 2.

The various field sizes were selected so that on both sides the chosen prefixes are now 96 bits short of the same MD5 block boundary. On both sides these 96 bit positions are reserved for the birthday bits of the chosen-prefix collision attack. On the legitimate side these 96 bits are part of the RSA modulus, on the rogue side they are part of an extension field of the type "Netscape Comment" described as "X.509 comment" in Table 3.

After the chosen-prefix collision has been constructed in Step 3, the certificates need to use a shared suffix. This part is determined by the legitimate certificate and simply copied to the rogue certificate, since on the rogue side this suffix is part of the "Netscape Comment" field and is ignored by all clients. On the legitimate side after the 3 near-collision blocks of 512 bits each, another 208 bits are used to complete a 2048-bit RSA modulus (determined in Step 4). This is followed by the RSA public exponent (the common value of 65537) and the X.509v3 extensions including the bit indicating that the legitimate certificate will be an end-user certificate (in Figure 3 denoted by "CA=FALSE").

Note that the legitimate certificate looks inconspicuous to the Certificate Authority, except maybe the long and weird requested domain name. In contrast, the rogue certificate looks very suspicuous under manual inspection: "Netscape Comment" fields are uncommon for CA certificates, and its weird structured contents that belongs to an end-user certificate is a big red flag. However, users rarely inspect certificates closely, and software clients simply ignore these suspicious elements.

Step 2. Prediction of serial number and validity period. Based on repeated certification requests submitted to the targeted commercial CA, it turned out that the validity period can very reliably be predicted as the period of precisely one year plus one day, starting exactly six seconds after a request is submitted. So, to control that field is quite easy: all one needs to do is select a validity period of the right length, and submit the legitimate certification request precisely six seconds before it starts.

Predicting the serial number is harder but not impossible. In the first place, it was found that the targeted commercial CA uses sequential serial numbers. Being able to predict the next serial number, however, is not enough: the construction of the collision

can be expected to take at least a day, before which the serial number and validity period have to be fixed, and only after which the to-be-signed part of the certificate will be entirely known. As a consequence, there will have been a substantial and uncertain increment in the serial number by the time the collision construction is finished. So, another essential ingredient of the construction was the fact that the CA's weekend workload is quite stable: it was observed during several weekends that the increment in serial number over a weekend does not vary a lot. This allowed to pretty reliably predict Monday morning's serial numbers on the Friday afternoon before. See Step 6 how the precise selected serial number and validity period was targeted.

Step 3. Computation of the collision. At this point both chosen prefixes have been fully determined so the chosen-prefix collision can be computed: first the birthday bits per side, followed by the calculation of 3 pairs of 512-bit near-collision blocks. The constraint of just 3 pairs of near-collision blocks causes the birthday search to be significantly more costly compared to the near-collision block construction, whereas normally these can be balanced against each other by allowing more near-collision block pairs. However, the available resources in the form of Arjen Lenstra's cluster of 215 PlayStation 3 (PS3) game consoles at EPFL was very suitable for the job. The entire calculation takes on average about a day on this cluster using suitable chosen-prefix collision parameters as described below.

When running Linux on a PS3, applications have access to 6 Synergistic Processing Units (SPUs), a general purpose CPU, and about 150MB of RAM per PS3. For the most costly phase, the birthday search phase, the 6×215 SPUs are computationally equivalent to approximately 8600 regular 32-bit cores, due to each SPU's 4×32-bit wide SIMD architecture. The other parts of the chosen-prefix collision construction (differential characteristic construction and near-collision block search) are much harder to implement efficiently on SPUs, but for these we were able to use the 215 PS3 CPUs effectively. With these resources, the choice for a much larger differential characteristic family (cf. Table 8 using $w = 5$) than normal still turned out to be acceptable despite the 1000-fold increase in the cost of the actual near-collision block construction.

We optimized the overall birthday search complexity for the plausible case that the birthday search takes $\sqrt{2}$ times longer than expected. Limited to 150MB per PS3, for a total of about 30GB, the choice was made to force $k = 8$ additional colliding bits in the birthday search (cf. Section 4.2).

Given the available timeline of a weekend, and that the calculation can be expected to take just a day, a number of chosen-prefixes were process sequentially, each corresponding to different serial numbers and validity periods (targetting both Monday and Tuesday mornings). Since the PS3 SPUs are solely used for the birthday phase, and the PS3 CPUs solely for the near-collision attack phase, several attempts could be pipelined. So, a near-collision block calculation on the CPUs would always run simultaneously with a birthday search on the SPUs for the 'next' attempt.

Step 4. Finishing the to-be-signed parts. At this point the legitimate and rogue sides collide under MD5, so that from here on only identical bits may be appended to both sides.

With $208 + 24 + 72 + 3 * 512 = 1840$ bits set, the remaining $2048 - 1840 = 208$ bits need to be set for the 2048-bit RSA modulus on the legitimate side. Because in the next step the RSA private exponent corresponding to the RSA public exponent is needed, the full factorization of the RSA modulus needs to be known, and the factors must be compatible with the choice of the RSA public exponent which can be achieved as follows. Let N be the 2048-bit integer consisting of the 1840 already determined bits of the RSA modulus-to-be, followed by 208 one bits. Select a 224-bit integer p at random until $N = a \cdot p + b$ with $a \in \mathbb{N}$ and $b < 2^{208}$, and keep doing this until both p and $q = \lfloor N/p \rfloor$ are prime and the RSA public exponent is coprime to $(p-1)(q-1)$. Once such primes p and q have been found, the number pq is the legitimate side's RSA modulus, the leading 1840 bits of which are already present in the legitimate side's to-be-signed part, and the 208 least significant bits of which are inserted in both to-be-signed parts.

To analyze the required effort somewhat more in general, 2^{k-208} integers of k bits (with $k > 208$) need to be selected on average for pq to have the desired 1840 leading bits. Since an ℓ-bit integer is prime with probability approximately $1/\log(2^\ell)$, a total of $k(2048 - k)2^{k-208}(\log 2)^2$ attempts may be expected before a suitable RSA modulus is found. The coprimality requirement is a lower order effect that is disregarded. Note that for $k(k - 2048)(\log 2)^2$ of the attempts the k-bit number p has to be tested for primality, and that for $(2048-k)\log 2$ of those q needs to be tested as well (on average, obviously). For $k = 224$ this turned out to be doable in a few minutes on a standard PC.

This completes the to-be-signed parts on both sides. Now it remains to be hoped that the legitimate part that actually will be signed corresponds, bit for bit, with the legitimate to-be-signed part that was constructed in this manner.

Step 5. Request certification of the legitimate to-be-signed part. Using the relevant information from the legitimate side's template, i.e., the subject Distinguished Name and the public key, a PKCS#10 Certificate Signing Request is prepared. The CA requires proof of possession of the private key corresponding to the public key in the request. This is done by signing the request using the private key – this is the sole reason that we need the RSA private exponent.

The targeted legitimate to-be-signed part contains a very specific validity period that leaves no choice for the moment at which the certification request needs to be submitted to the CA. Just hoping that at that time the serial number would have precisely the predicted value is unlikely to work, so a somewhat more elaborate approach is used. About half an hour before the targeted submission moment, the same request is submitted, and the serial number in the resulting certificate is inspected. If it is already too high, the entire attempt is abandoned. Otherwise, the request is repeatedly submitted, with a frequency depending on the gap that may still exist between the serial number received and the targeted one, and taking into account possible certification requests by others. In this way the serial number is slowly nudged toward the right value at the right time.

Various types of accidents were experienced, such as another CA customer 'stealing' the targeted serial number just a few moments before the attempt to get it, thereby wasting that weekend's calculations. But, after the fourth weekend it worked as planned to get an actually signed part that exactly matched a predicted legitimate to-be-signed part.

Given the perfect match between the actually signed part and the hoped for one, and the MD5 collision between the latter and the rogue side's to-be-signed part, the MD5-based digital signature present in the legitimate certificate as provided by the commercial CA is equally valid for the rogue side. To finish the rogue CA certificate it suffices to copy the digital signature to the right spot in the rogue CA certificate.

3.3 The FLAME supermalware

In response to the rogue CA construction, various authorities explicitly disallowed MD5 in digital signatures. For instance, the CA/Browser Forum adopted Baseline Requirements for CAs in 2011,[4] and Microsoft updated its Root CA Program in 2009.[5] Nevertheless, surprisingly MD5 was not completely removed for digital signatures within Microsoft, which came to light in 2012 with the discovery [Lab12a, Lab12b] of the FLAME super malware for espionage in the Middle-East supposedly in a joint US-Israel effort [Pos12]. This was a highly advanced malware that could collect a vast collection of data including files, keyboard inputs, screen contents, microphone, webcam and network traffic, sometimes even triggered by use of specific applications of interest [Lab12a, Lab12b]. In contrast to normal malware, FLAME infections occured with surgical precision to carefully selected targets, which helped evading detection possibly since 2007 [Lab12a].

Of cryptanalytic interest is one of its means of infection [Sot12]: FLAME used WPAD (Web Proxy Auto-Discovery Protocol) to register itself as a proxy for the domain `update.windows.com` to launch Man-in-the-Middle attacks for Windows Update on other computers on the local network. By forcing a fall-back from the secure `HTTPS` protocol to the unauthenticated `HTTP` protocol, FLAME was able to push its own validly signed Windows Update patches to other Windows machines. It was a huge surprise finding that FLAME actually was in possession of its own illegitimately signed Windows Update patch to infect other machines, after inspection it became clear it was constructed with a collision attack [Mic12]. It was discovered that MD5 signatures were still being used for licensing purposes in their Terminal Server Licensing Service, which automatically generated MD5-based signed certificates that lead up to the Microsoft Root CA. Using a collision attack enabled the attackers to convert a signed license certificate into a validly signed code signing certificate that was accepted for Windows Updates by all versions of Windows [Mic12]. Below we go more into detail on two very interesting aspects: the colliding certificates and the used chosen-prefix collision attack.

The colliding certificates. Normally Windows Update executables are signed by code-signing certificate in a dedicated Windows Update certificate chain under the Microsoft Root certificate. However, the FLAME code-signing certificate is connected to a completely different part of the PKI tree leading up to the same Root certificate, as illustrated below in Figure 1.

As described by Trail-of-Bits [Sot12], there is an automated process whereby a customer-side running License Server generates a private key and a X.509 certificate

[4] `https://cabforum.org/wp-content/uploads/Baseline_Requirements_V1.pdf`
[5] `https://technet.microsoft.com/en-us/library/cc751157.aspx`

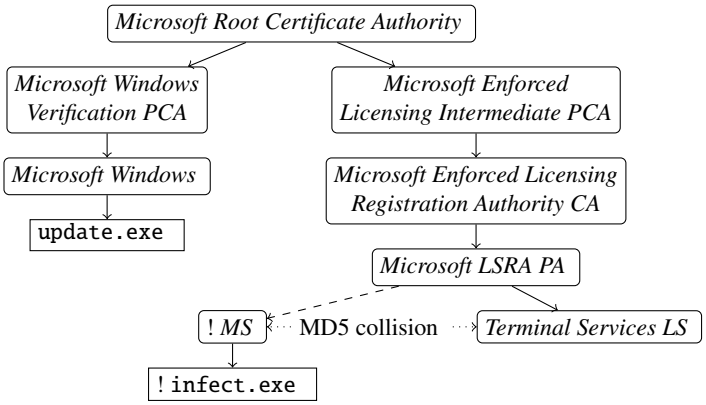

Fig. 1: Windows Update and FLAME Certificate tree

signing request with: (1) customer information, (2) machine id, (3) corresponding public key. A Microsoft activation server then returns a certificate with an MD5-based signature from "Microsoft LSRA PA" (see above). This signed certificate has the following properties:

- No identifying information, except CommonName="Terminal Services LS"
- Fixed validity period until Feb 19, 2012
- No Extended Key Usage restrictions
  $\Rightarrow$ restrictions inherited from CA certificate, which includes code signing
- special critical Microsoft Hydra X.509 extension
  marked *critical* thus will be rejected if not understood by Crypto library

The private key and certificate can be recovered by the customer, and as it supports code-signing and chains up to the Microsoft Root CA, it was found out that any "Terminal Services LS"-certificate is sufficient to sign Windows Update executables for Windows XP. For Windows Vista and Windows 7, however, the critical Hydra extension causes such signed executables to be rejected. It was for this reason that a chosen-prefix collision attack was employed such that the "MS"-certificate was obtained that could sign Windows Update executables for all Windows versions at that time.

From the FLAME malware the full "MS" certificate has been analyzed, however the colliding "Terminal Services LS" certificate has been lost. Nevertheless, this was sufficient to estimate the layout of the colliding certificates depicted below.

The chosen-prefix collision attack. Once the FLAME certificate structure was figured out and it was clear a chosen-prefix collision attack was used, it was immediately thought the publicly available chosen-prefix collision implementation [Ste09b] was used. With a novel technique called *counter-cryptanalysis* [Ste13a] we succeeded in recovering all near-collision block data of the lost legitimate certificate from the FLAME rogue certificate and thereby the differential characteristics used for each near-collision attack. Surprisingly, it became immediately clear that these did not match

	lost legitimate license cert (A)	rogue WU signing cert (B)
prefix	? serial number A ? validity period A CN="Terminal Services LS" start ? 4096-bit RSA key B "	serial number B validity period B CN="MS" 2048-bit RSA key A start issuerUniqueID field to end
collision	collision data A	collision data B
common suffix	tail of RSA key critical Hydra extension	<copy from A> <copy from A>

Table 4: The to-be-signed parts of FLAME colliding certificates

the known chosen-prefix collision attacks: the message block differences used were $\delta m_4 = \delta m_1 4 = 2^{31}$ and $\delta m_{11} = \pm 2^{15}$, which so far were only used Wang et al.'s first MD5 (identical prefix) collision attack [WY05] and improvements thereof. This made it obvious that neither the HashClash implementation [Ste09b] or even the best known chosen-prefix collision attack [SSA$^+$09] were used, but a yet unknown new variant chosen-prefix collision attack developed by world-class cryptanalists. A closer look on the attack details is described in section 4.4.

3.4 Colliding PGP keys with different identities

Recently, Leurent et al.[LP20] demonstrated a pair of PGP keys with different identities that collide under SHA-1. This implies that any certification of one key can be transferred to the other key, directly undermining the web of trust of PGP keys. This was quickly fixed by deprecating SHA-1 in GnuPG in the modern branch, while a legacy branch remains unaltered and insecure. The main structure of these colliding files is depicted in Table 5, one of the notable features is that the collision data precedes any user data. The produced collision is thus reusable for many attacks.

	Key A	Key B	bytes
prefix	header A	header B	4
	timestamp A	timestamp B	4
	key A size: 8192 bits	key B size: 6144 bits	3
collision	collision data A	collision data B	757
common suffix	<copy from B>	remaining RSA key	16
	<copy from B>	PGP image header	24
	<copy from B>	tiny JPEG image	181
	remaining RSA key	<copy from A>	51
	PGP user id	<copy from A>	30
	<copy from B>	signature trailer	18

Table 5: PGP Key collision [LP20]

Their attack reused the GPU implementation of the first identical-prefix collision for SHA-1 [SBK+17] with some refinements and adjustments as the basis for all near-collision attacks, but the main improvement lies in the near-collision attack strategy presented earlier by the same authors [LP19]. Their strategy exploits two facts:

1. Each near-collision attack can essentially produce many output differences with almost the same probability
2. For each chaining value difference there are many distinct choices which differences each near-collision attack eliminates

They propose the represent all possible choices in a weighted graph, using which they can compute for each near-collision attack which set of possible differences to target given the remaining chaining value difference so far. By targetting many output differences the cost of near-collision attacks drops significantly, where the last block that has to eliminate the remaining chaining value difference has only one target output difference is the most costly. In total this first SHA-1 chosen-prefix collision has an estimated cost of $2^{63.4}$ SHA-1 compression functions, which could be executed very economically for the bargain price of 75k$ by renting a cheap GPU farm (originally built for blockchain mining).

3.5 Transcript collision attacks on TLS, IKE and SSH

While consensus was reached among certification authorities and software vendors to stop issuing and accepting MD5-based signatures and certificates, MD5 remained in use in many popular security applications. Bhargavan and Leurent [BL16] demonstrated in 2016 meet-in-the-middle attacks on the key exchange protocols underlying various version of TLS, IPsec and SSH. In particular, they show how to impersonate TLS-1.2 clients, TLS-1.3 servers and IKEv2 initiators, as well as downgrade TLS-1.1 and SSHv2 connections to use weak ciphers. In these meet-in-the-middle attacks the two honest parties will observe different protocol execution transcripts. However online chosen-prefix collision attacks can be exploited to ensure transcript hashes used for mutual authentication collide, which allows authentication messages can be successfully forwarded.

Most of these attacks are quite involved and depend on the low-level details of those protocols, however we show a simple example man-in-the-middle attack in Figure 2 for the Sign-and-MAC protocol [Kra03]. After the first message m_1 by A, the attacker has to find an online collision $h = H(m_1|m'_2) = H(m'_1|m_2)$, where it also needs to correctly predict the first response m_2 by B after receiving m'_1. At that point the attacker can successfully setup a shared session key both with A ($g^{xy'}$) and with B ($g^{x'y}$). In the final step the transcripts are signed by both parties in each connection for mutual authentication. As the to-sign transcript hashes collide, the signatures are valid for both sessions. The attacker can simply forward those signatures between A and B to pretend A and B have an authenticated private channel.

In contrast to most other chosen-prefix collision attacks described here, these transcript collision attacks have to be carried out online. This places heavy constraints on the wallclock time available to compute the collision attack for a successful outcome.

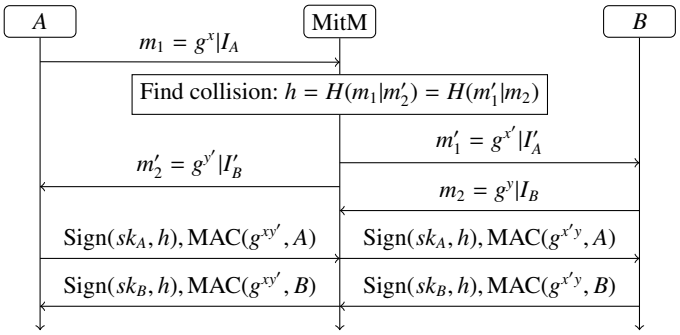

Fig. 2: Man-in-the-middle attack on the Sign-and-Mac protocol

This authors report special modifications to the HashClash software [Ste09b] to avoid large memory-disk transfers required for the interaction between the different programs by merging those programs, and by using all possible precomputations. These modifications reportedly enabled them to reduce the wallclock time of computing chosen-prefix collisions on a 48-core machine from at least three hours to only one hour. Collisions scale rather well with amount of computational power available, although it remains unclear how practically feasible it is to bring this down to say within a minute. For most online attacks even a minute delay, let alone an one hour delay, in the key exchange is very troublesome, and unthinkable with direct human interaction as is for internet browsers. But it appears that some TLS client software are willing to wait indefinitely on the key exchange as long as regularly messages are being sent such as warning alerts, so such TLS clients that remain unsupervised may be susceptible.

3.6 Nostradamus attack on hash-based commitments

Kelsey and Kohno [KK06] presented a Nostradamus attack to first commit to a hash value, after which a document containing any message of one's choice can be constructed that matches the committed hash value with lower cost than the generic preimage attack cost. The method applies to any Merkle-Damgård hash function, such as MD5, that given a chaining value and a suffix produces another chaining value. Depected in Figure 3 and omitting details involving message lengths and padding, the idea is to build a tree of 2^d chaining values that all lead to a single root chaining value h_{root} and then to commit to a hash value h_{commit} computed by hashing some chosen suffix starting from h_{root}. The tree is a complete binary tree and is calculated from its leaves up to the root, so the root h_{root} will be the last value calculated. This is done in such a way that each node of the tree is associated with a chaining value along with a suffix that together hash to the chaining value associated with the node's parent. Thus, two siblings have chaining values values and suffixes that collide under the hash function. The chaining values at the leaves may be arbitrarily chosen but are, preferably, all different.

After committing to h_{commit}, given a prefix msg of one's choice one performs a brute-force search for a suffix x such that hashing $msg||x$ results in the chaining value of one of the leaves of the tree. Appending the message suffixes one encounters on the path from

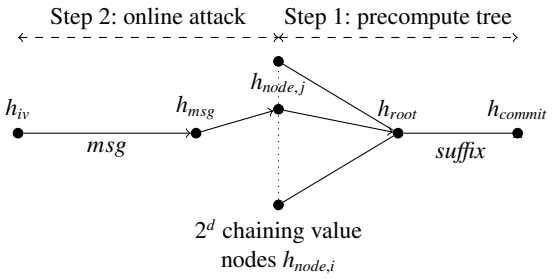

Fig. 3: Nostradamus attack

that leave to the root, results in a final message with the desired prefix and committed hash value.

For MD5, however, it remains far from feasible to carry out the entire construction in practice as it requires massive birthday searches to find the necessary 2^{d+1} collisions. However, there are two different variants that have been demontrated in practice.

First, for very small d there is a variant that *is* feasible, as then the construction of the tree can be done efficiently by using chosen-prefix collisions to construct sibling node suffixes based on their chaining values. An example of this variant is discussed in section 3.8 involving 12 PDF documents.

Another actually practical approach is discussed in section 3.14. This variant uses a large 2^d multicollision created from d sequential collision attacks, in combination with file format exploit techniques to allow these to visually show 2^d different shown messages when rendered. The examples in section 3.14 use this approach to craft *hashquines*: documents that show their own MD5 hash.

3.7 Colliding documents

In [DL05] it was shown how to construct a pair of PostScript files that collide under MD5, but that display different messages when viewed or printed. These constructions use identical-prefix collisions and thus the only difference between the colliding files is in the generated collision bit strings. See also [GIS06] for similar constructions for other document formats. These constructions have to rely on the presence of both messages in each of the colliding files and on macro-functionalities of the document format used to show either one of the two messages.

This can be improved with chosen-prefix collisions as one message per colliding document suffices and macro-functionalities are no longer required. For example, using a document format that allows insertion of color images (such as Microsoft Word or Adobe PDF), inserting one message per document, two documents can be made to collide by appending carefully crafted color images after the messages. A short one pixel wide line will do – for instance hidden inside a layout element, a company logo, or a nicely colored barcode – and preferably scaled down to hardly visible size (or completely hidden from view, as possible in PDF). An extension of this construction is presented in the paragraphs below and set forth in detail in Section 3.8.

3.8 PDF files and the Nostradamus attack

In [SLdW12], Stevens, Lenstra and de Weger describe a detailed construction for PDF files that also demonstrates a variant scenario of the Nostradamus attack [KK06]. In the original Nostradamus attack one first commits to a certain hash value, and afterwards for any message constructs a document that not only contains that message but that also has the committed hash value. This attack is at this point in time not feasible for MD5 in its full generality, however it is easily doable if a limited size message space has been defined upfront.

Suppose there are messages $m_1, m_2, \ldots, m_r$, then using $r - 1$ chosen-prefix collisions one can construct r suffixes $s_1, s_2, \ldots, s_r$ such that the r documents $d_i = m_i \| s_i$ all have the same hash. After committing to the common hash, afterwards any of the r documents $d_1, d_2, \ldots, d_r$ can be shown, possibly to achieve some malicious goal. The other documents will remain hidden and their prefix parts, i.e., the m_i-parts, cannot be derived from the single published document or from the common hash value. However, given the structure of PDF documents it is not entirely straightforward to insert different chosen-prefix collision blocks, while keeping the parts following those blocks identical in order to maintain the collision as described below.

In [SLdW12], we succeed in constructing 12 different PDF documents with a common MD5 hash value, where each document predicts a different outcome of the 2008 US presidential elections. The common MD5 hash value of the 12 colliding PDF documents is

`3D5 15 DEAD 7AA16560ABA3E9DF05CBC80`.

See [SLdW07a] for the actual PDF documents, one of which correctly predicted the outcome one year before the elections took place.

A PDF document is built up from the following four consecutive parts: a fixed header, a part consisting of an arbitrary number of numbered "objects", an object lookup table and, finally, a trailer. The trailer specifies the number of objects, which object is the unique root object (containing the root of the document content tree) and which object is the info object (containing the document's meta information such as authors and title etc.), and contains a filepointer to the start of the object lookup table.

Given a file containing a PDF document, additional objects can be inserted, as long as they are added to the object lookup table and the corresponding changes are made to the number of objects and the filepointer in the trailer. If these are not referenced by the document content tree anywhere then they will be ignored by any PDF processor. The format is mostly text based, with the exception of binary images where collision data can be safely hidden. The used template for inserted image objects is given in Table 6, which offers sufficient room to hide 11 chosen-prefix collisions with 12 message blocks each (1 birthday search block and 11 near-collision blocks). Any binary image is put between single line-feed characters (ASCII code 10) and the result is encapsulated by the keywords `stream` and `endstream`. The keyword `/Length` must specify the byte length of the image equal to $3 \times \mathit{width} \times \mathit{height}$, since for uncompressed images each pixel requires three bytes ('`RGB`'). The object number (`42` in the example object header) must be set to the next available object number.

Part	Contents
object header	`42 0 obj`
image header	`<< /ColorSpace /DeviceRGB /Subtype /Image`
image size	`/Length 9216 /Width 64 /Height 48 /BitsPerComponent 8`
image contents	`>> stream...endstream`
object footer	`endobj`

Table 6: An example numbered image object in the PDF format

When constructing colliding PDF files they must be equal after the collision-causing data. The object lookup tables and trailers for all files must therefore be the same. This was achieved as follows:

- Because all documents must have the same number of objects, dummy objects are inserted where necessary.
- Because all root objects must have the same object number, they can be copied if necessary to objects with the next available object number.
- The info objects are treated in the same way as the root objects.
- To make sure that all object lookup tables and filepointers are identical, the objects can be sorted by object number and if necessary padded with spaces after their `obj` keyword to make sure that all objects with the same object number have the same file position and byte length in all files.
- Finally, the object lookup tables and trailers need to be adapted to reflect the new situation – as a result they should be identical for all files.

Although this procedure works for basic PDF files such as produced using pdflatex, it should be noted that the PDF document format allows additional features that may cause obstructions.

Given r LaTeX files with the desired subtle differences (such as names of r different candidates), r different PDF files are produced using a version of LaTeX that is suitable for our purposes (cf. above). In all these files a binary image object with a fixed object number is then inserted, and the approach sketched above is followed to make the lookup tables and trailers for all files identical. Since this binary image object is present but not used in the PDF document, it remains hidden from view in a PDF reader. To ensure that the files are identical after the hidden image contents, their corresponding objects were made the last objects in the files. This then leads to r chosen prefixes consisting of the leading parts of the PDF files up to and including the keyword `stream` and the first line-feed character. After determining $r - 1$ chosen-prefix collisions resulting in r collision-causing appendages, the appendages are put in the proper binary image parts, after which all files are completed with a line-feed character, the keywords `endstream` and `endobj`, and the identical lookup tables and trailers.

Note that the `Length` etc. fields have to be set before collision finding, and that the value of `Length` will grow logarithmically with r and linearly in the number of near-collision blocks one is aiming for.

3.9 Colliding executables & software integrity checking

In [Kam04] and [Mik04] it was shown how any existing MD5 collision, such as the ones originally presented by Xiaoyun Wang at the Crypto 2004 rump session, can be abused to mislead integrity checking software that uses MD5. A similar application of colliding executables, using freshly made collisions, was given on [Sel06]. All these results use identical-prefix collisions and, similar to the colliding PostScript application mentioned earlier, differences in the colliding inputs are used to construct deviating execution flows.

However, digitally signed executables contain the signature itself, so in order to craft a collision for the hash input to the digital signature a slight modification is necessary. Didier Stevens [Ste09a] describes how this can be achieved for Microsoft Authenticode, which uses a special "Attribute Certificate Table" section in the file containing the certificate chain and digital signatures. When computing the signature hash input one hashes the byte string of the executable file except for three parts that are cut out: a 4-byte checksum value, an 8-byte pointer to Attribute Certificate Table, and the Attribute Certificate table itself.

Chosen-prefix collisions allow a more elegant approach, since common operating systems ignore bit strings that are appended to executables: the programs will run unaltered, which is demonstrated in [SLdW07b]. Furthermore, using tree-structured chosen-prefix collision appendages as in Section 3.8, any number of executables can be made to have the same MD5 hash value or MD5-based digital signature.

One can imagine two executables: a 'good' one (say Word.exe) and a 'bad' one (the attacker's Worse.exe). A chosen-prefix collision for those executables is computed, and the collision-causing bit strings are appended to both executables. The resulting altered file Word.exe, functionally equivalent to the original Word.exe, can be offered to a code signing program such as Microsoft's Authenticode and receive an 'official' MD5-based digital signature. This signature will then be equally valid for the attacker's Worse.exe, and the attacker might be able to replace Word.exe by his Worse.exe (renamed to Word.exe) on the appropriate download site. This construction affects a common functionality of MD5 hashing and may pose a practical threat. It also allows people to get many executables signed at once at the cost of getting a single such executable signed, bypassing verification of any kind (e.g., authenticity, quality, compatibility, non-spyware, non-malware) by the signing party of the remaining executables.

3.10 Digital forensics

In digital forensics, there are two main uses for cryptographic hash function: file identification and integrity verification, for which MD5 is unsuitable. Contrary to the abundant evidence in the literature as described in this survey, a position paper has been published in 2019 that MD5 remains appropriate for integrity verification and file identification in the field of digital forensics by the *Scientific Working Group on Digital Evidence* (SWDGE) [Sci19]. This document claims that

> *"It is appropriate to use both MD5 and SHA1 for integrity verification provided the hash is securely stored or recorded in the examination documentation. This*

will prevent an individual from substituting a different file and its hash. This is true for all hash algorithms. Since there are no preimage attacks on any of the four hashing algorithms discussed, the only way to manipulate the evidence without detection is to do it before it is hashed."

It fails to acknowledge that collisions can be crafted in advance after which manipulation is undetectable with a non-collision resistant hash function. Note that actual manipulation of the evidence in custody may not be necessary for collisions to be problematic.

File identification. So-called *hash sets* are used to quickly identify known files. For example, when a hard disk is seized by law enforcement officers, they may compute the hashes of all files on the disk, and compare those hashes to hashes in existing hash sets: a whitelist (for known harmless files such as operating system and other common software files) and a blacklist (for previously identified harmful files). Only files whose hashes do not occur in either hash set have to be inspected further. A useful feature of this method of recognizing files is that the file name itself is irrelevant, since only the content of the file is hashed.

MD5 is a popular hash function for this application. Examples are NIST's National Software Reference Library Reference Data Set[6] and the US Department of Justice's Hashkeeper application[7].

A conceivable, and rather obvious, attack on this application of hashes is to produce a harmless file (e.g., an innocent picture) and a harmful one (e.g., an illegal picture), and insert collision blocks that will not be noticed by common application software or human viewers. In a learning phase the harmless file might be submitted to the hash set and thus the common hash may end up on the whitelist. The harmful file will be overlooked from then on.

Integrity verification. When devices are seized by law enforcement officers, they may compute the hashes of all files on the disk to digitally certify the contents. These can later be used in court to verify the chain of custody and to cryptographically verify the integrity of the evidence that it has not been tampered with.

Using document collisions, one can envision a potential criminal ensures all criminating files collide with harmless ones. He could try to dismiss the use of the incriminating files in court, by presenting the harmless files that match the hash values of the incriminating files and casting doubt on the chain of custody. For instance, in 2005 the Australian Hornsby Local Court dismissed a vehicle speeding case after the prosecutor could not provide evidence that the speed camera image hashed with MD5 had not been tampered with[Sch05].

Another threat scenario of using colliding files, one harmless and one incriminating, is planting the harmless file on a victim's device later to be swapped by the incriminating one. Both cases may result in law enforcement presenting the incriminating file and the defendent presenting the harmless file. From a purely cryptographic hash point of view,

[6] `http://www.nsrl.nist.gov/`

[7] `http://www.usdoj.gov/ndic/domex/hashkeeper.htm`

these cases are indistinguishable when using a non-collision resistant hash function such as MD5.

3.11 Peer to peer software

Hash sets such as for digital forensics are also used in peer to peer software. A site offering content may maintain a list of pairs (file name, hash). The file name is local only, and the peer to peer software uniquely identifies the file's content by means of its hash. Depending on which hash is used and how the hash is computed such systems may be vulnerable to a chosen-prefix attack. Software such as eDonkey and eMule use MD4 to hash the content in a two stage manner: the identifier of the content $c_1\|c_2\|\ldots\|c_n$ is MD4(MD4(c_1)$\|\ldots\|$MD4(c_n)), where the chunks c_i are about 9 MB each. One-chunk files, i.e., files not larger than 9 MB, are most likely vulnerable; whether multi-chunk files are vulnerable is open for research. Chosen-prefix collision attacks have not been presented against MD4. But as MD4 is a simpler version of MD5, an adaptation of MD5's attacks should result in an attacks on MD4 that are considerably faster compared to MD5's attacks.

3.12 Content addressed storage

Content addressed storage is a means of storing fixed content at a physical location of which the address is directly derived from the content itself. For example, a hash of the content may be used as the file name. See [PD05] for an example. Clearly, chosen-prefix collisions can be used by an attacker to fool such storage systems, e.g., by first preparing colliding pairs of files, by then storing the harmless-looking first one, and later overwriting it with the harmful second one.

3.13 Polyglots: multi-format collisions

As file formats typically start with a specific fixed header that indicate the file format used, two colliding files of different file formats are difficult to construct using identical-prefix collisions. Using chosen-prefix collisions, one can take any two files and make them collide, one only has to correctly hide the collision bits for the file format's renders. The best known example of 4 files with distinct file formats (PNG image, PDF document, MP4 video and PE executable) has been crafted by Ange Albertini.[8] In this case, it is a *reusable* collision where the chosen prefixes only contain the file format headers, and do not cover any file content. A simple program by Albertini uses the preconstructed 4-way collision to convert given PNG, PDF, MP4 and PE files into 4 colliding files with the same hash.

3.14 Hashquines: embedding the MD5 hash in documents

Another variant on the Nostradamus attack (see section 3.6) that is feasible for short messages is as follows. The goal is to build a 2^d multi-collision crafted by d sequential

[8] `https://github.com/corkami/collisions`

identical-prefix collisions, where each of the 2^d files all having the same hash renders a different message. The precise method depends strongly on the used file format. Later, after some event, one can choose which particular file out of the 2^d colliding files to show that matches the committed hash. One possible choice for the rendered message is the hash value itself, such documents that render their own hash are called *hashquines*. Hashquines have been demonstrated for the following file formats in the Poc||GTFO journal [Lap17]: GIF, PDF, Nintendo NES ROM and PostScript.

In the case of executables it would be almost trivial to include these d sequential identical-prefix collisions, read out the chosen 'bit' b_i of collision i ($b_i = 0$ if collision data A is used, $b_i = 1$ if collision data B is used) and map the resulting d-bit string to a to-be-rendered message in the desired message space. After the complete file has been crafted one can commit to its hash.

A more common strategy that works for several file formats is to place collisions inside comment fields, where one of the message block differences affects the comment length. In this manner, one can force different behaviours for parsers: either it parses a short comment followed by some content to render, or it parses a long comment that skips over the content. This idea for the JPEG file format is depicted in Figure 4, where 16 same size JPEG images are placed into one file. Each of the 16 JPEG images renders to one of the 16 hexidecimal symbols (0,..,9,a,..,f), and is prefixed by a comment whose length is controlled by a collision. In this case, long comments force the parser to skip over an image, and only the first short comment results in the next JPEG image data to be rendered. Any following JPEG images will be ignored. An MD5 hash consists of 32 hexidecimal symbols, so a PDF hashquine can be constructed by constructing 32 such JPEG images inside a PDF document. And once the PDF file MD5 hash is known, the 32 JPEG images can be altered (without changing the MD5 hash) such that together they render the MD5 hash.

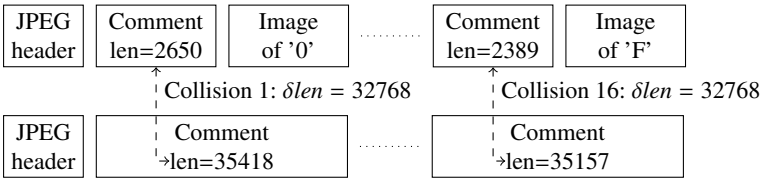

Fig. 4: JPEG multi-collision: 16 different images, each prefixed with a comment whose length can be controlled with a collision. Only the first image that is not skipped over by a comment is rendered.

4 MD5 Collision attacks

In the sections below, we first give a brief mathematical description of MD5 itself. Then we discuss the details of several interesting chosen-prefix collision attacks: the currently best chosen-prefix collision attack [SSA+09] in section 4.2, a single block

variant [SSA+09] in section 4.3 and FLAME's attack by Nation States exposed by Counter-cryptanalysis [Ste13a, FS15] in section 4.4.

4.1 MD5

The hash function MD5 was designed by Ron Rivest in 1992 with a 128-bit output hash and is built on the Merkle-Damgård framework using its compression function *MD5Compress*.

1. *Padding.* Pad the message with a '1'-bit followed by the minimal number of '0'-bits to pad to a bitlength of 448 modulo 512. Append the bitlength of the original unpadded message as a 64-bit little-endian integer.
2. *Partitioning.* The padded message is partitioned into k consecutive 512-bit blocks $M_1, \ldots, M_k$.
3. *Processing.* MD5's chaining values CV_i are 128-bits long consisting of four 32-bit words a_i, b_i, c_i, d_i. For the initial value CV_0, these are fixed public values:

   $$(a_0, b_0, c_0, d_0) = (\texttt{0x67452301}, \texttt{0xEFCDAB89}, \texttt{0x98BADCFE}, \texttt{0x10325476}).$$

   For $i = 1, \ldots, k$, the chaining value is updated:

   $$CV_i = (a_i, b_i, c_i, d_i) = \mathit{MD5Compress}(CV_{i-1}, M_i).$$

4. *Output.* The resulting hash value is CV_k expressed as the concatenation of the hexadecimal byte strings of a_k, b_k, c_k, d_k (using little-endian).

MD5's compression function *MD5Compress* internally uses 32-bit *words* that are both used as integers modulo 2^{32} for addition (least significant bit is right-most bit), and as 32-bit strings for bitwise operations: AND ($\wedge$), OR ($\vee$), XOR ($\oplus$), NOT ($\overline{X}$), left/right rotation by n-bits (left: $X^{\lll n}$, right: $X^{\ggg n}$).

MD5Compress takes as input a 128-bit chaining value CV split into 4 words (a, b, c, d) and a 512-bit message block B split into 16 words $m_0 \| \cdots \| m_{15}$. We use an unrolled description of *MD5Compress* that initializes four state words with the chaining values $(Q_0, Q_{-1}, Q_{-2}, Q_{-3}) = (b, c, d, a)$ and performs 64 rounds for $t = 0, \ldots, 63$ in which it computes a next state word Q_{t+1}:

$$\begin{aligned} F_t &= f_t(Q_t, Q_{t-1}, Q_{t-2}) \\ T_t &= F_t + Q_{t-3} + AC_t + W_t \\ R_t &= T_t^{\lll RC_t} \\ Q_{t+1} &= Q_t + R_t, \end{aligned}$$

where $AC_t = \lfloor 2^{32} \cdot |\sin(t+1)| \rfloor$ and W_t, $f_t(X, Y, Z)$ and RC_t are given in Table 7. At the end, *MD5Compress* returns $(a + Q_{61}, b + Q_{64}, c + Q_{63}, d + Q_{62})$.

Step	W_t	$f_t(X, Y, Z)$	RC_t
$0 \le t < 16$	m_t	$(X \wedge Y) \oplus (\overline{X} \wedge Z)$	$(7, 12, 17, 22)_{[t \bmod 4]}$
$16 \le t < 32$	$m_{(1+5t) \bmod 16}$	$(Z \wedge X) \oplus (\overline{Z} \wedge Y)$	$(5, 9, 14, 20)_{[t \bmod 4]}$
$32 \le t < 48$	$m_{(5+3t) \bmod 16}$	$X \oplus Y \oplus Z$	$(4, 11, 16, 23)_{[t \bmod 4]}$
$48 \le t < 64$	$m_{(7t) \bmod 16}$	$Y \oplus (X \vee \overline{Z})$	$(6, 10, 15, 21)_{[t \bmod 4]}$

Table 7: MD5 round constants and boolean functions

t	δQ_t	δF_t	δW_t	δT_t	δR_t	RC_t
31	$\mp 2^{p-10 \bmod 32}$					
32	0					
33	0					
34	0	0	$\pm 2^{p-10 \bmod 32}$	0	0	16
$35 - 60$	0	0	0	0	0	$\cdot$
61	0	0	$\pm 2^{p-10 \bmod 32}$	$\pm 2^{p-10 \bmod 32}$	$\pm 2^p$	10
62	$\pm 2^p$	0	0	0	0	15
63	$\pm 2^p$	0	0	0	0	21
64	$\pm 2^p$ $+ \sum_{\lambda=0}^{w'} s_\lambda 2^{p+21+\lambda \bmod 32}$					

Table 8: Family of partial differential characteristics using $\delta m_{11} = \pm 2^{p-10 \bmod 32}$, where $s_0, \ldots, s_{w'} \in \{-1, 0, +1\}$ and $w' = \min(w, 31 - p)$ for a fixed $w \ge 0$. Interesting values for the parameter w are between 2 and 5.

4.2 MD5 chosen-prefix collision attack

The 2009 chosen-prefix collision construction used for the rogue CA certificate (c.f. section 3.2) remains the best known attack. The attack has several parameters that can be adjusted. First, the maximum number of allowed near-collision blocks is denoted by r and can be used to trade-off between birthday search time complexity and the cost of finding the r near-collision blocks. Second, k defines the birthday search space (its size is $64 + k$) and the birthday iteration function and can be used to trade-off between birthday search time complexity, birthday search memory complexity and average number of required near-collisions per birthday collision. Third, w defines the family of differential characteristics that can be used to construct the near-collision blocks and is the number of bit positions where arbitrary bit differences are allowed. It can be used to trade-off between the average number of required near-collision blocks per birthday collision and the cost of finding the r near-collision blocks.

The attack is based on the family of partial differential characteristics described in Table 8 using a single message block bit difference $\delta m_{11} = \pm 2^i$ where both the sign and bit position i can be varied. This results in a difference $(\delta a, \delta b, \delta c, \delta d)$ to be added to the chaining value difference, where $\delta a = 0$, $\delta d = \delta c = \pm 2^{i+10 \bmod 32}$, while δb consists of a fixed bit difference $\pm 2^{i+10 \bmod 32}$ as well as arbitrarily chosen differences on bit positions $i + 21, \ldots, i + 21 + w$ (modulo 32).

Birthday search. As described in section 2, the chosen-prefix collision attack starts with a tailored birthday search that results in a chaining value difference that can be eliminated by a sequence of near-collision attacks. In this case the search space V and iteration function f depend on the integer parameter $k \in \{0, 1, 2, \ldots, 32\}$. More precisely, first pad prefixes P and P' with arbitrary suffixes S_{r} and S'_{r} such that the total bit lengths are equal, and $64 + k$ bits short of a multiple of the blocksize 512 bits. Then given $k \in \{0, \ldots, 32\}$, let B and B' be the last $512 - (64 + k)$ bits of padded prefixes $P \| S_{\mathrm{r}}$ and $P' \| S'_{\mathrm{r}}$, respectively. Then V and f are defined as follows:

$$V = \mathbb{Z}_{2^{32}} \times \mathbb{Z}_{2^{32}} \times \mathbb{Z}_{2^k},$$

$$f(x, y, z) = (a, c - d, c - b \bmod 2^k) \text{ where}$$

$$(a, b, c, d) = \begin{cases} MD5Compress(CV_{n-1}, B \| x \| y \| z) & \text{if } x \bmod 2 = 0; \\ MD5Compress(CV'_{n-1}, B' \| x \| y \| z) & \text{if } x \bmod 2 = 1. \end{cases}$$

The birthday collision, however, needs to satisfy several additional conditions that cannot be captured by V, f, or k: the prefixes associated with x and y in a birthday collision $f(x) = f(y)$ must be different, and the required number of pairs of near-collision blocks may be at most r when allowing differential characteristics with parameter w. The probability that a birthday collision satisfies all requirements depends not only on the choice of r and w, but also on the value for k, and is denoted by $p_{r,k,w}$. As a consequence, on average $1/p_{r,k,w}$ birthday collisions have to be found.

$k = 0$	$w = 0$			$w = 1$			$w = 2$			$w = 3$		
r	p	C_{tr}	M	p	C_{tr}	M	p	C_{tr}	M	p	C_{tr}	M
16	5.9	35.27	1MB	1.75	33.2	1MB	1.01	32.83	1MB	1.	32.83	1MB
15	7.2	35.92	1MB	2.39	33.52	1MB	1.06	32.86	1MB	1.	32.83	1MB
14	8.71	36.68	1MB	3.37	34.01	1MB	1.27	32.96	1MB	1.04	32.84	1MB
13	10.45	37.55	1MB	4.73	34.69	1MB	1.78	33.22	1MB	1.2	32.93	1MB
12	12.45	38.55	1MB	6.53	35.59	1MB	2.78	33.71	1MB	1.66	33.16	1MB
11	14.72	39.68	2MB	8.77	36.71	1MB	4.34	34.5	1MB	2.61	33.63	1MB
10	17.28	40.97	11MB	11.47	38.06	1MB	6.54	35.6	1MB	4.18	34.42	1MB
9	20.16	42.4	79MB	14.62	39.64	2MB	9.38	37.02	1MB	6.46	35.56	1MB
8	23.39	44.02	732MB	18.21	41.43	21MB	12.88	38.76	1MB	9.52	37.09	1MB
7	26.82	45.73	8GB	22.2	43.43	323MB	17.02	40.83	9MB	13.4	39.02	1MB
6	31.2	47.92	161GB	26.73	45.69	8GB	21.78	43.22	241MB	18.14	41.4	20MB
5	35.	49.83	3TB	31.2	47.92	161GB	27.13	45.89	10GB	23.74	44.2	938MB
4							34.	49.33	2TB	30.19	47.42	81GB

The columns p, C_{tr} and M denote the values of $-\log_2(p_{r,k,w})$, $\log_2(C_{tr}(r, k, w))$ and the minimum required memory M such that $C_{coll}(r, k, w, M) \leq C_{tr}(r, k, w)$, respectively.

Table 9: Expected birthday search costs for $k = 0$

Assuming that M bytes of memory are available and that a single birthday trail requires 28 bytes of storage (namely 96 bits for the start- and endpoint each, and 32 for

the length), this leads to the following expressions for the birthday search costs [vW99]:

$$C_{tr}(r,k,w) = \sqrt{\frac{\pi \cdot 2^{64+k}}{2 \cdot p_{r,k,w}}}, \quad C_{coll}(r,k,w,M) = \frac{2.5 \cdot 28 \cdot C_{tr}(r,k,w)}{p_{r,k,w} \cdot M}.$$

For $M = 70/p_{r,k,w}$ as given in the last column of Table 9, the two costs are equal, and the overall expected birthday search costs becomes $2C_{tr}(r,k,w)$. However, in practice it is advisable to choose M considerably larger. For $\epsilon \leq 1$, using $M = 70/(p_{r,k,w} \cdot \epsilon)$ bytes of memory results in $C_{coll} \approx \epsilon \cdot C_{tr}$ and the expected overall birthday search cost is about $(1+\epsilon) \cdot C_{tr}(r,k,w)$ MD5 compressions.

Near-collision construction algorithm. The birthday search results in a chaining value difference δCV_n of the form $(0, \delta b, \delta c, \delta c)$. Let $\delta c = \sum_i k_i 2^i$ and $\delta b - \delta c = \sum_i l_i 2^i$, where $(k_i)_{i=0}^{31}$ and $(l_i)_{i=0}^{31}$ are signed binary digit expansions in Non-Adjacent Form (NAF). [9] If $\delta c \neq 0$, let i be such that $k_i \neq 0$. Using a differential characteristic from Table 8 with $\delta m_{11} = -k_i 2^{i-10 \bmod 32}$ one can eliminate the difference $k_i 2^i$ in δc and δd and simultaneously change δb by

$$k_i 2^i + \sum_{\lambda = i+21 \bmod 32}^{i+21+w' \bmod 32} l_\lambda 2^\lambda,$$

where $w' = \min(w, 31-i)$. Here one needs to be careful that each non-zero l_λ is eliminated only once in the case when multiple i-values allow the elimination of l_λ. Doing this for all k_i that are non-zero in the NAF of δc results in a difference vector $(0, \widehat{\delta b}, 0, 0)$ where $\widehat{\delta b}$ may be different from δb, and where the weight $w(NAF(\widehat{\delta b}))$ may be smaller or larger than $w(NAF(\delta b))$. More precisely, $\widehat{\delta b} = \sum_{\lambda=0}^{31} e_\lambda l_\lambda 2^\lambda$, where $e_\lambda = 0$ if there exist indices i and j with $0 \leq j \leq \min(w, 31-i)$ such that $k_i = \pm 1$ and $\lambda = 21+i+j \bmod 32$ and $e_\lambda = 1$ otherwise.

The bits in $\widehat{\delta b}$ can be eliminated as follows. Let $(\widehat{l_i})_{i=0}^{31} = NAF(\widehat{\delta b})$ and let j be such that $\widehat{l_j} = \pm 1$ and $j - 21 \bmod 32$ is minimal. Then the difference $\sum_{i=j}^{j+w'} \widehat{l_i} 2^i$ with $w' = \min(w, 31 - (j - 21 \bmod 32))$ can be eliminated from $\widehat{\delta b}$ using $\delta m_{11} = 2^{j-31 \bmod 32}$, which introduces a new difference $2^{j-21 \bmod 32}$ in δb, δc and δd. This latter difference is eliminated using $\delta m_{11} = -2^{j-31 \bmod 32}$, which then leads to a new difference vector $(0, \overline{\delta b}, 0, 0)$ with $w(NAF(\overline{\delta b})) < w(NAF(\widehat{\delta b}))$. The process is repeated until all differences have been eliminated. We refer to [SLdW07c, SSA+09] for details on the construction of the near-collision attacks themselves, including the differential characteristic construction algorithm and collision search algorithms, and to [Ste09b] for an implementation of the entire chosen-prefix collision attack.

Complete differential characteristics can be constructed with an average total complexity equivalent to roughly 2^{35} MD5 compressions. For small $w = 0, 1, 2$ and differential characteristics based on Table 8, finding a near-collision block pair requires on average roughly the equivalent of 2^{34} MD5 compressions. Combined with the construction of the differential characteristics, this leads to a rough overall estimate of about

[9] Note: $NAF(a)$ for integer a denotes the Non-Adjacent Form of a, i.e., the unique signed binary digit expansion $(a_i)_{i=0}^{n}$ with $a_i \in \{-1, 0, 1\}$ and $a = \sum_i a_i 2^i$ where no two non-zero digits are adjacent.

$2^{35.6}$ MD5 compressions to find a single pair of near-collision blocks for this chosen-prefix collision attack. The optimal parameters are given as $w = 2$, $k = 0$ and $r = 9$, for which the birthday search cost is about 2^{37} MD5 compressions and constructing the $r = 9$ pairs of near-collision blocks costs about $2^{38.8}$ MD5 compressions, leading to the claimed total complexity of about $2^{39.1}$ MD5 compressions.

4.3 MD5 Single-block chosen-prefix collision attack

It is even possible to construct a chosen-prefix collision using only a single pair of near-collision blocks using a slightly different strategy [SSA+09]. Together with 84 birthday bits, the chosen-prefix collision-causing appendages are only $84 + 512 = 596$ bits long. This attack is based on a large family of differential characteristics that enables for a corresponding large set of chaining value differences to be eliminated using a single pair of near-collision blocks.

Instead of using the family of differential characteristics based on $\delta m_{11} = \pm 2^i$, this attack reuses the fastest known collision attack for MD5 and varies the last few steps to find a large family of differential characteristics depicted in Table 10. Specifically, by varying those last steps and allowing the collision finding complexity to grow by a factor of about 2^{26}, this results in a set $\mathcal{S}$ of about $2^{23.3}$ different $\delta CV = (\delta a, \delta b, \delta c, \delta d)$ of the form $\delta a = -2^5$, $\delta d = -2^5 + 2^{25}$, $\delta c = -2^5 \bmod 2^{20}$ that can be eliminated. Such δCVs can be found using an 84-bit birthday search with step function $f : \{0,1\}^{84} \to \{0,1\}^{84}$ of the form

$$f(x) = \begin{cases} \phi(MD5Compress(CV, B\|x) + \widehat{\delta CV}) & \text{for } \tau(x) = 0 \\ \phi(MD5Compress(CV', B'\|x)) & \text{for } \tau(x) = 1, \end{cases}$$

where $\widehat{\delta CV}$ is of the required form, $\tau : x \mapsto \{0,1\}$ is a balanced partition function and $\phi(a,b,c,d) \mapsto a\|d\|(c \bmod 2^{20})$. There are $2^{128-84} = 2^{44}$ possible δIHV values of this form, of which only about $2^{23.3}$ are in the allowed set $\mathcal{S}$. It follows that a birthday collision $f(x) = f(x')$ has probability $p = 2^{23.3}/(2^{44} \cdot 2) = 2^{-21.7}$ to be useful, where the additional factor 2 stems from the fact that different prefixes are required, i.e., $\tau(x) \neq \tau(x')$.

A useful birthday collision can be expected after $\sqrt{\pi 2^{84}/(2p)} \approx 2^{53.2}$ MD5 compressions and requires approximately 400MB of storage. The average complexity of finding the actual near-collision blocks is bounded by about $2^{14.8+26} = 2^{40.8}$ MD5 compressions and negligible compared to the birthday complexity. Thus the overall complexity is approximately $2^{53.2}$ MD5 compressions.

4.4 FLAME's MD5 chosen-prefix collision attack

As described in section 3.3, using counter-cryptanalysis all near-collision block pairs were recovered and thus the differential characteristics which proved that the used chosen-prefix collision was a yet-unknown variant attack [Ste13a]. After this initial discovery, Fillinger and Stevens performed a in-depth analysis wherein they reverse engineered this yet unknown chosen-prefix collision attack [FS15]. The FLAME chosen-prefix collision attack differs significantly on several aspects from public research:

t	δQ_t	δF_t	δW_t	δT_t	δR_t	RC_t
26	-2^8					
27	0					
28	0					
29	0	0	2^8	0	0	9
30 – 33	0	0	0	0	0	·
34	0	0	2^{15}	2^{15}	2^{31}	16
35	2^{31}	2^{31}	2^{31}	0	0	23
36	2^{31}	0	0	0	0	4
37	2^{31}	2^{31}	2^{31}	0	0	11
38 – 46	2^{31}	2^{31}	0	0	0	·
47	2^{31}	2^{31}	2^8	2^8	2^{31}	23
48	0	0	0	0	0	6
49	0	0	0	0	0	10
50	0	0	2^{31}	0	0	15
51 – 59	0	0	0	0	0	·
60	0	0	2^{31}	2^{31}	-2^5	6
61	-2^5	0	2^{15}	2^{15}	2^{25}	10
62	$-2^5 + 2^{25}$	0	2^8	2^8	2^{23}	15
63	$-2^5 + 2^{25} + 2^{23}$	$2^5 - 2^{23}$	0	$2^5 - 2^{23}$	$2^{26} - 2^{14}$	21
64	$-2^5 + 2^{25} + 2^{23} + 2^{26} - 2^{14}$					

Partial differential characteristic for $t = 29, \ldots, 63$ using message differences $\delta m_2 = 2^8$, $\delta m_4 = \delta m_{14} = 2^{31}$, $\delta m_{11} = 2^{15}$. The probability that it is satisfied is approximately $2^{-14.5}$. It leads to a identical-prefix collision attack of approximated complexity 2^{16} MD5 compressions.

Table 10: Partial differential characteristic for fast near-collision attack

- The near-collision attack strategy to eliminate chaining value differences ($\delta a, \delta b, \delta c, \delta d$) using four blocks. Where δa and δd are essentially fixed and together have only three bit differences. Each of the four blocks focuses on a specific region of bit positions of δb and δc to cancel, while using the freedom to arbitrarily affect bit differences in bit position regions handled by later blocks. The last two blocks are able to cancel all bit differences in δb on bit positions [0-4,13-19,27-31], but cancel only the last remaining bit difference in δc which is likely fixed in advance. The first two blocks cancel essentially all bit differences in δc, they also affect δb but seemingly in a random manner: any difference added to δb is allowed as long as the last two blocks are still able to effectively cancel them.
- The differential characteristic construction method follows a variant meet-in-the-middle approach. Unlike [SLdW07c] that uses many lower characteristics and many upper characteristics and then tries to combine pairs into a complete valid differential characteristic, the FLAME differential characteristics are fixed from δQ_6 up to δQ_{60}. This signifies that it uses only one upper characteristic and tries to connect many lower characteristics to it. The designers have chosen to have bit differences at all bit positions of Q_6, which may maximize the success probability. In the end,

this appears to be significantly slower, as fixing δQ_6 in this manner implies a significant factor increase in the number of pairs that have to be tried, compared to [SLdW07c]. In comparison, the HashClash implementation is able to find replacement differential characteristics for FLAME's differential characteristics in a matter of seconds on a desktop computer, where the number of resulting bit conditions are also significantly lower: e.g. the replacement first characteristic has only 276 bit conditions versus 328 for FLAME's characteristic.

– The near-collision block search makes use of known collision search speed-up techniques as actual bit values match many necessary bit conditions, which is unlikely to occur by chance. Noticable is that not all speed-up techniques that could be applied are visible in the actual blocks, as for some the necessary bit conditions are not satisfied. This is not evidence that not all speed-up techniques are used. One possible explanation is that some speed-up techniques are used dynamically, depending on whether their necessary bit conditions are set. This would increase degrees of freedom, but only limited impact on the near-collision block search complexity.

Fillinger and Stevens reconstruct a parametrized family of chosen-prefix collision attacks that should contain the FLAME collision attack. The precise parameters used for FLAME cannot be recovered with enough certainty. But it was possible to determine the minimal cost parameters leading to estimated cost $C_{min} = 2^{46.6}$, and the minimal cost parameters that are consistent with the observed data leading to estimated cost $C_{flame} = 2^{49.3}$. This is significantly higher than the estimated cost 2^{39} of the currently best known MD5 chosen-prefix collision attack [SSA+09]. However it was also observed that FLAME's attack is more suitable to be implemented on massively parallel hardware such as GPUs, which has practical benefits.

Bibliography

[BC04] Eli Biham and Rafi Chen, *Near-collisions of SHA-0*, CRYPTO 2004 (Santa Barbara, CA, USA) (Matthew Franklin, ed.), LNCS, vol. 3152, Springer, Heidelberg, Germany, August 15–19, 2004, pp. 290–305.

[BL16] Karthikeyan Bhargavan and Gaëtan Leurent, *Transcript collision attacks: Breaking authentication in TLS, IKE and SSH*, NDSS 2016 (San Diego, CA, USA), The Internet Society, February 21–24, 2016.

[Dam90] Ivan Damgård, *A design principle for hash functions*, CRYPTO'89 (Santa Barbara, CA, USA) (Gilles Brassard, ed.), LNCS, vol. 435, Springer, Heidelberg, Germany, August 20–24, 1990, pp. 416–427.

[dB94] Bert den Boer and Antoon Bosselaers, *Collisions for the compressin function of MD5*, EUROCRYPT'93 (Lofthus, Norway) (Tor Helleseth, ed.), LNCS, vol. 765, Springer, Heidelberg, Germany, May 23–27, 1994, pp. 293–304.

[DL05] Magnus Daum and Stefan Lucks, *Attacking hash functions by poisoned messages, "the story of Alice and her boss"*, June 2005, `https://web.archive.org/web/20160713130211/http://th.informatik.uni-mannheim.de:80/people/lucks/HashCollisions/`.

[DR06] Christophe De Cannière and Christian Rechberger, *Finding SHA-1 characteristics: General results and applications*, ASIACRYPT 2006 (Shanghai, China) (Xuejia Lai and Kefei Chen, eds.), LNCS, vol. 4284, Springer, Heidelberg, Germany, December 3–7, 2006, pp. 1–20.

[FS15] Max Fillinger and Marc Stevens, *Reverse-engineering of the cryptanalytic attack used in the Flame super-malware*, ASIACRYPT 2015, Part II (Auckland, New Zealand) (Tetsu Iwata and Jung Hee Cheon, eds.), LNCS, vol. 9453, Springer, Heidelberg, Germany, November 30 – December 3, 2015, pp. 586–611.

[GIS06] Max Gebhardt, Georg Illies, and Werner Schindler, *A note on the practical value of single hash collisions for special file formats*, Sicherheit, LNI, vol. P-77, GI, 2006, pp. 333–344.

[JP07] Antoine Joux and Thomas Peyrin, *Hash functions and the (amplified) boomerang attack*, CRYPTO 2007 (Santa Barbara, CA, USA) (Alfred Menezes, ed.), LNCS, vol. 4622, Springer, Heidelberg, Germany, August 19–23, 2007, pp. 244–263.

[Kam04] Dan Kaminsky, *MD5 to be considered harmful someday*, Cryptology ePrint Archive, Report 2004/357, 2004, `https://eprint.iacr.org/2004/357`.

[KK06] John Kelsey and Tadayoshi Kohno, *Herding hash functions and the Nostradamus attack*, EUROCRYPT 2006 (St. Petersburg, Russia) (Serge Vaudenay, ed.), LNCS, vol. 4004, Springer, Heidelberg, Germany, May 28 – June 1, 2006, pp. 183–200.

[Kli05] Vlastimil Klima, *Finding MD5 collisions on a notebook PC using multi-message modifications*, Cryptology ePrint Archive, Report 2005/102, 2005, https://eprint.iacr.org/2005/102.

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

[Kra03] Hugo Krawczyk, *SIGMA: The "SIGn-and-MAc" approach to authenticated Diffie-Hellman and its use in the IKE protocols*, CRYPTO 2003 (Santa Barbara, CA, USA) (Dan Boneh, ed.), LNCS, vol. 2729, Springer, Heidelberg, Germany, August 17–21, 2003, pp. 400–425.

[Lab12a] CrySyS Lab, *skywiper (a.k.a. flame a.k.a. flamer): A complex malware for targeted attacks*, Laboratory of Cryptography and System Security, Budapest University of Technology and Economics, May 31, 2012.

[Lab12b] Kaspersky Lab, *The flame: Questions and answers*, Securelist blog, May 28, 2012.

[Lap17] Manul Laphroaig (ed.), *Pastor laphroaig screams high five to the heavens as the whole world goes under*, PoC||GTFO, vol. 0x14, Tract Association of POC||GTFO and Friends, 2017.

[LdW05] Arjen K. Lenstra and Benne de Weger, *On the possibility of constructing meaningful hash collisions for public keys*, ACISP 05 (Brisbane, Queensland, Australia) (Colin Boyd and Juan Manuel González Nieto, eds.), LNCS, vol. 3574, Springer, Heidelberg, Germany, July 4–6, 2005, pp. 267–279.

[LP19] Gaëtan Leurent and Thomas Peyrin, *From collisions to chosen-prefix collisions application to full SHA-1*, EUROCRYPT 2019, Part III (Darmstadt, Germany) (Yuval Ishai and Vincent Rijmen, eds.), LNCS, vol. 11478, Springer, Heidelberg, Germany, May 19–23, 2019, pp. 527–555.

[LP20] Gaëtan Leurent and Thomas Peyrin, *Sha-1 is a shambles - first chosen-prefix collision on sha-1 and application to the pgp web of trust*, Cryptology ePrint Archive, Report 2020/014, 2020, https://eprint.iacr.org/2020/014.

[Mer90] Ralph C. Merkle, *One way hash functions and DES*, CRYPTO'89 (Santa Barbara, CA, USA) (Gilles Brassard, ed.), LNCS, vol. 435, Springer, Heidelberg, Germany, August 20–24, 1990, pp. 428–446.

[MHP09] Cameron McDonald, Philip Hawkes, and Josef Pieprzyk, *Differential path for SHA-1 with complexity $O(2^{52})$*, Cryptology ePrint Archive, Report 2009/259, 2009, https://eprint.iacr.org/2009/259.

[Mic12] Microsoft, *Flame malware collision attack explained*, Security Research & Defense, Microsoft TechNet Blog, June 6, 2012.

[Mik04] Ondrej Mikle, *Practical attacks on digital signatures using MD5 message digest*, Cryptology ePrint Archive, Report 2004/356, 2004, https://eprint.iacr.org/2004/356.

[MRR07] Florian Mendel, Christian Rechberger, and Vincent Rijmen, *Update on sha-1*, Rump session of CRYPTO 2007, 2007, http://rump2007.cr.yp.to/09-rechberger.pdf.

[PD05] Robert Primmer and Carl D'Halluin, *Collision and preimage resistance of the centera content address*, Technical Report, 2005, EMC Corporation.

[Pos12] The Washington Post, *U.S., Israel developed flame computer virus to slow Iranian nuclear efforts, officials say*, Ellen Nakashima, Greg Miller and Julie Tate, June 2012.

[Rog06] Phillip Rogaway, *Formalizing human ignorance: Collision-resistant hashing without the keys*, Cryptology ePrint Archive, Report 2006/281, 2006, https://eprint.iacr.org/2006/281.

[SBK+17] Marc Stevens, Elie Bursztein, Pierre Karpman, Ange Albertini, and Yarik Markov, *The first collision for full SHA-1*, CRYPTO 2017, Part I (Santa Barbara, CA, USA) (Jonathan Katz and Hovav Shacham, eds.), LNCS, vol. 10401, Springer, Heidelberg, Germany, August 20–24, 2017, pp. 570–596.

[Sch05] Bruce Schneier, *The MD5 defense*, Schneier on Security blog, august 2005, https://www.schneier.com/blog/archives/2005/08/the_md5_defense.html.

[Sci19] Scientific Working Group on Digital Evidence, *SWGDE position on the use of MD5 and SHA1 hash algorithms in digital and multimedia forensics*, September 2019.

[Sel06] Peter Selinger, 2006, http://www.mathstat.dal.ca/~selinger/md5collision/.

[SLdW07a] Marc Stevens, Arjen Lenstra, and Benne de Weger, *Predicting the winner of the 2008 US presidential elections using a sony playstation 3*, 2007, http://www.win.tue.nl/hashclash/Nostradamus/.

[SLdW07b] ———, *Vulnerability of software integrity and code signing*, 2007, http://www.win.tue.nl/hashclash/SoftIntCodeSign/.

[SLdW07c] Marc Stevens, Arjen K. Lenstra, and Benne de Weger, *Chosen-prefix collisions for MD5 and colliding X.509 certificates for different identities*, EUROCRYPT 2007 (Barcelona, Spain) (Moni Naor, ed.), LNCS, vol. 4515, Springer, Heidelberg, Germany, May 20–24, 2007, pp. 1–22.

[SLdW12] ———, *Chosen-prefix collisions for MD5 and applications*, IJACT **2** (2012), no. 4, 322–359.

[Sot12] Alex Sotirov, *Analyzing the MD5 collision in flame*, SummerCon 2020, New York, USA, June 2012.

[SSA+09] Marc Stevens, Alexander Sotirov, Jacob Appelbaum, Arjen K. Lenstra, David Molnar, Dag Arne Osvik, and Benne de Weger, *Short chosen-prefix collisions for MD5 and the creation of a rogue CA certificate*, CRYPTO 2009 (Santa Barbara, CA, USA) (Shai Halevi, ed.), LNCS, vol. 5677, Springer, Heidelberg, Germany, August 16–20, 2009, pp. 55–69.

[Ste06] Marc Stevens, *Fast collision attack on MD5*, Cryptology ePrint Archive, Report 2006/104, 2006, https://eprint.iacr.org/2006/104.

[Ste07] ———, *On collisions for md5*, June 2007.

[Ste09a] Didier Stevens, 2009, http://blog.didierstevens.com/2009/01/17/.

[Ste09b] Marc Stevens, *Github: Project hashclash - MD5 & SHA-1 cryptanalysis*, June 2009, https://github.com/cr-marcstevens/hashclash.

[Ste12] ———, *Attacks on hash functions and applications*, Ph.D. thesis, Leiden University, June 2012.

[Ste13a] ———, *Counter-cryptanalysis*, CRYPTO 2013, Part I (Santa Barbara, CA, USA) (Ran Canetti and Juan A. Garay, eds.), LNCS, vol. 8042, Springer, Heidelberg, Germany, August 18–22, 2013, pp. 129–146.

[Ste13b] ———, *New collision attacks on SHA-1 based on optimal joint local-collision analysis*, EUROCRYPT 2013 (Athens, Greece) (Thomas Johansson and Phong Q. Nguyen, eds.), LNCS, vol. 7881, Springer, Heidelberg, Germany, May 26–30, 2013, pp. 245–261.

[vW99] Paul C. van Oorschot and Michael J. Wiener, *Parallel collision search with cryptanalytic applications*, Journal of Cryptology **12** (1999), no. 1, 1–28.

[WY05] Xiaoyun Wang and Hongbo Yu, *How to break MD5 and other hash functions*, EUROCRYPT 2005 (Aarhus, Denmark) (Ronald Cramer, ed.), LNCS, vol. 3494, Springer, Heidelberg, Germany, May 22–26, 2005, pp. 19–35.

[WYY05a] Xiaoyun Wang, Andrew C. Yao, and Frances Yao, *Cryptanalysis on SHA-1*, NIST Cryptographic Hash Workshop, 2005, `http://csrc.nist.gov/groups/ST/hash/documents/Wang_SHA1-New-Result.pdf`.

[WYY05b] Xiaoyun Wang, Yiqun Lisa Yin, and Hongbo Yu, *Finding collisions in the full SHA-1*, CRYPTO 2005 (Santa Barbara, CA, USA) (Victor Shoup, ed.), LNCS, vol. 3621, Springer, Heidelberg, Germany, August 14–18, 2005, pp. 17–36.

[XLF08] Tao Xie, Fanbao Liu, and Dengguo Feng, *Could the 1-MSB input difference be the fastest collision attack for MD5?*, Cryptology ePrint Archive, Report 2008/391, 2008, `https://eprint.iacr.org/2008/391`.