Security/Efficiency Tradeoffs for Permutation-Based Hashing

Phillip Rogaway[1] and John Steinberger[2]

[1] Department of Computer Science, University of California, Davis, USA
[2] Department of Mathematics, University of British Columbia, Canada

Abstract. We provide attacks and analysis that capture a tradeoff, in the ideal-permutation model, between the speed of a permutation-based hash function and its potential security. We show that any $2n$-bit to n-bit compression function will have unacceptable collision resistance it makes fewer than three n-bit permutation invocations, and any $3n$-bit to $2n$-bit compression function will have unacceptable security if it makes fewer than five n-bit permutation invocations. Any rate-α hash function built from n-bit permutations can be broken, in the sense of finding preimages as well as collisions, in about $N^{1-\alpha}$ queries, where $N = 2^n$. Our results provide guidance when trying to design or analyze a permutation-based hash function about the limits of what can possibly be done.

1 Introduction

Overview. Consider the problem of constructing a cryptographic hash function where, for reasons of speed, assurance, or minimalism, you've decided to base your design on an off-the-shelf blockcipher, say AES, with an $n = 128$ bit blocksize and a small, fixed set of keys. To keep things modular, you've decided to first build a $3n$-bit to $2n$-bit compression function from your n-bit permutations $\pi_1, \ldots, \pi_k$. You plan to prove your construction sound in the ideal-permutation model, where the adversary has black-box access to the forward and backwards direction for each π_i.

Perhaps surprisingly, the design problem just described is extremely challenging. If you write a construction down, chances are good that, after a while, you'll find an efficient attack. It's quite unlikely you'll find an easy proof. At least this was our experience, and over a period of many months.

In this paper we partially explain *where* the design difficulty is coming from. Basically, the problem is that *it costs a surprisingly large number of permutation invocations to buy a reasonable level of security.* In particular, compressing $3n$ bits to $2n$ bits needs at least *five* permutation invocations just to break the birthday bound of $N^{0.5}$ queries (where $N = 2^n$) that motivates having a double-length construction in the first place. And even with five permutations there is *still* going to be a collision-finding attack that uses about $N^{0.6}$ queries, which isn't all that great.

In prior work, Black, Cochran, and Shrimpton [1] showed that any rate-1 iterated hash function whose compression function uses a single permutation call must be insecure in the ideal-permutation model.[1] In the present work, the Black *et al.* result is seen as a point on a continuum: while one permutation call is not enough, more and more calls buys you, potentially, better and better security. Concretely, we exhibit a quantifiable tradeoff between the number of permutation calls and the effectiveness of a corresponding attack. The attack's effectiveness diminishes rather slowly with the number of permutation calls.

The problem of constructing a cryptographic hash function from a fixed-key blockcipher dates to Preneel, Govaerts, and Vandewalle [8]. They explain the utility of this problem and specify a family of solutions with inverse rates of 4–8. For the concrete parameters they suggest, a compression function mapping 310 bits to 256 bits using four calls to 64-bit permutations, our own pigeonhole-birthday attack (Theorem 2) implies that an adversary will probably have the information it needs to construct a collision after making just two million queries. While this doesn't mean that there's a computationally efficient way to *find* the desired collision, it does mean that, for the stated parameters, one can't possibly prove a decent security bound in the random-permutation model.

We want to emphasize at the outset that this paper is about attacks, not constructions or their security proofs. It remains an intriguing open question if, for every choice of parameters, there *is* a construction whose provable security matches that given by our attacks. Our guess is that the answer is *yes*, which would mean that the results of this paper are tight.

Our results and their interpretation. Let us now summarize our results one-by-one. First we look at the collision resistance of a permutation-based compression function. We show that if a compression function maps mn bits to rn bits using k calls to n-bit permutations—a signature we abbreviate as $m \xrightarrow{k} r$, eliding n—then an adversary will be able to find a collision using some[2] $N^{1-(m-0.5r)/k}$ queries, where, again and throughout, $N = 2^n$. In particular, a $2 \xrightarrow{2} 1$ compression function can be broken with about $N^{1-(2-0.5)/2} = N^{1/4}$ queries, which is unacceptably few, while a $3 \xrightarrow{4} 2$ compression function can be broken in about about $N^{1-(3-1)/4} = N^{1/2}$ queries, which, for a double-length construction, is again too few.

Our bounds suggest a qualitative difference in behavior between the $m \xrightarrow{k} 1$ (single-length) and the $m \xrightarrow{k} 2$ (double-length) settings: in the first case $k = 3$ permutations is enough to potentially achieve the optimal security of $N^{1/2}$ queries, while in the second case no number of permutation calls can ever achieve the optimal security of N queries. It has recently been shown that one *can* asymptotically achieve the optimal security of $N^{1/2}$ queries with a $2 \xrightarrow{3} 1$ compression function [9], one of the rare choices of parameters for which a $m \xrightarrow{k} r$ construction is known to have a security bound matching that of our attacks.

[1] The *rate* of a permutation-based hash function is α if it processes αn bits worth of data with each n-bit permutation invocation. The inverse rate $\beta = 1/\alpha$ is therefore the number of permutation calls used per n bits of input.

[2] In summarizing our results we omit distracting multiplicands or addends that have a second-order effect.

Next we put compression functions aside and look at collision resistance for a full-fledged permutation-based hash function $H\colon \{0,1\}^* \to \{0,1\}^{rn}$. We show that if the rate of the hash function is α then an adversary can find collisions with about $N^{1-\alpha}$ queries. In particular, rate-1 hash functions are completely insecure, as already discovered by Black *et al.* for the special case of iterated hash functions using a single permutation call per iteration. In addition, a rate-1/2 double-length hash function ($r = 2$) will admit an $N^{1/2}$-query attack. As this is what one expects from a single-length construction, the conclusion is that a double-length construction must have a rate of less than 1/2.

We also look at the preimage resistance of permutation-based compression functions and hash functions. In the former case, a preimage for an $m \xrightarrow{k} r$ construction can be found in about $N^{1-(m-r)/k}$ queries. In particular, preimages can be found in any $2 \xrightarrow{3} 1$ design with about $N^{2/3}$ queries. (Happily, the $2 \xrightarrow{3} 1$ construction we mentioned asymptotically matches this bound [9].) So while collision-resistance can be "as good as a random function" with a $2 \xrightarrow{3} 1$ design, no such design can be comparably good with respect to preimage resistance. For a full-fledged rate-α hash function, a preimage can be found in about $N^{1-\alpha}$ queries, which is, rather oddly, the same as for collision resistance.

In a somewhat different spirit, Section 8 of this paper considers the number of bits that a permutation-based compression function must keep in memory in order to be collision resistant. We show that an $m \to r$ compression function must, at some point during its computation, keep strictly more than mn bits in memory, or else it will suffer from devastating attacks. If we imagine that the compression function is built from n-bit *wires* connecting the permutations, then the compression function must, at some point, maintain at least $m + 1$ active wires to have any hope for collision resistance.

Appendix A sketches a generalization of the attack of Black *et al.* Theirs is a collision attack on permutation-based iterated hash functions that use a single permutation call per iteration; here we adapt it to the case where k permutation calls are made per iteration. The attack is only applicable to iterated hash functions, and our version of it uses a heuristic assumption, but the bound is slightly better than that of our attack for an arbitrary hash function.

2 The Model

Consider a compression function $H\colon \{0,1\}^{mn} \to \{0,1\}^{rn}$ built from black-box n-bit permutations, where $m > r \geq 1$ and $n \geq 1$. Let us assume that for H to process its mn-bit input requires making k calls, in order, to permutations $\pi_1, \ldots, \pi_k\colon \{0,1\}^n \to \{0,1\}^n$. Then H necessarily takes the form illustrated in Fig. 1, for some sequence of functions $f_1, \ldots, f_k$, g. Along with permutations $\pi_1, \ldots, \pi_k\colon \{0,1\}^n \to \{0,1\}^n$, functions $f_i\colon \{0,1\}^{imn} \to \{0,1\}^n$ ($i \in [1..k]$) and $g\colon \{0,1\}^{(i+1)mn} \to \{0,1\}^{rn}$ define H. In general, we do not require anything of $f_1, \ldots, f_k$, g beyond their having the specified domain and range.

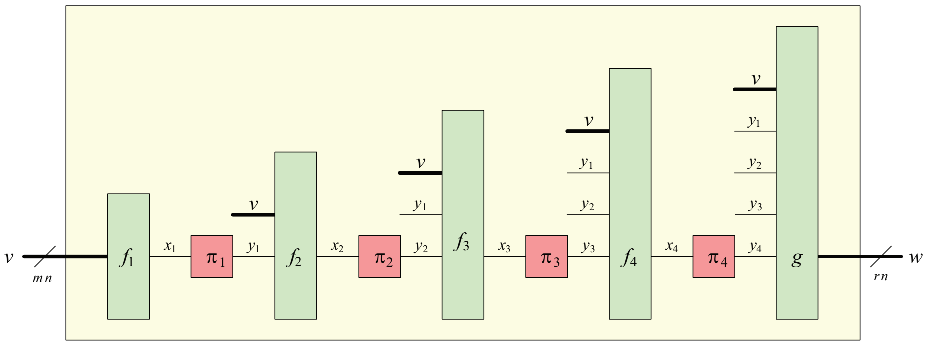
Because $\pi_1, \ldots, \pi_k$ are always called in the order π_1 and then π_2 and so forth, up to π_k, we call the model just described the *fixed-order* model. It includes designs where the permutations $\pi_1, \ldots, \pi_k$ are unrelated—the *distinct-permutation* setting—and designs where a single permutation π $(= \pi_1 = \cdots = \pi_k)$ is always called—the *single-permutation* setting. It does not include the case where the identity of the permutation (ie, which π_i is used at each step) is data dependent. This restriction turns out not to be so significant—more on that in just a bit.

Let H be a fixed-order compression function, notation as above, and let $\mathcal{A}$ be an adversary with access to oracles $\pi_1, \ldots, \pi_k$ (and, in principle, their inverses—only that this isn't needed in any of our attacks). The *advantage* of $\mathcal{A}$ in finding collisions in H is the probability that $\mathcal{A}$ asks a sequence of queries such that there exist distinct inputs $v, v' \in \{0,1\}^{mn}$ for which the adversary has asked all necessary queries to compute $H(v)$ and $H(v')$. This probability is over the adversary's coins and over uniform permutation oracles $\pi_1, \ldots, \pi_k$. (This sentence assumes the distinct-permutation setting. More generally, select a single random permutation to model each distinct π_i.) Note that we do not insist that the adversary actually output a collision: we assert that it wins if a computationally-unbounded adversary *could* compute a collision from what it knows. It is true that this makes the attacks less "realistic" than if we had paid attention to the attacker's time and required it to print out its collision. But since our main goal is to understand the limits of what is provably secure in the random-permutation model, we can ignore time and adopt a liberal notion of adversarial success.

As mentioned already, one can generalize the fixed-order model by letting the compression function choose which permutation to invoke at each step: in Fig. 1, add in a line 3.5 saying $j \leftarrow e_i(v, y_1, \ldots, y_{i-1})$, and use j, not i, as the subscript for π at line 4. This *no-fixed-order* model was employed by Black, Cochran, and Shrimpton [1]. We ourselves prefer the fixed-order model, and assume it for quantitative results. Philosophically, letting permutation selection vary according to the data being hashed would make permutation-based hashing conceptually coincide with blockcipher-based hashing, contrary to the point of our investigation. More pragmatically, good lower bounds in the (simpler) fixed-order setting are already enough to imply good lower bounds in the (more complex) no-fixed-order setting. To see this, note that if H is a no-fixed-order compression function that makes k permutation calls, then there's a functionally identical fixed-order compression function H' that makes k^2 calls: H' just queries its k permutations in a round-robin fashion. Because of this, lower-bounds applicable to (the fixed-order) H' are inherited by (the no-fixed-order) H if one simply replaces each k by k^2. Since we are always thinking of k as a small constant, the quantitative change in bounds is not so significant. In particular, every qualitative conclusion that we draw in this paper is an accurate interpretation of our results for the fixed-order model and the no-fixed-order model, too.

3 The Trivial Attacks

We begin by acknowledging two trivial but nonetheless significant attacks on any permutation-based compression function, the *exhaustion attack* and the *birthday*



```
1  algorithm $H(v)$
2  for $i \leftarrow 1$ to $k$ do
3      $x_i \leftarrow f_i(v,\ y_1, \ldots, y_{i-1})$
4      $y_i \leftarrow \pi_i(x_i)$
5  $w \leftarrow g(v,\ y_1, \ldots, y_k)$
6  return $w$
```

Fig. 1. Illustration and definition for a permutation-based compression function. Regarding $\pi_1, \ldots, \pi_k$ as oracles, functions $f_1, \ldots, f_k$ and g define the scheme, which maps an mn-bit input v to an rn-bit output w.

attack. The former attack asks all kN possible queries, where $N = 2^n$. At that point the hash of *every* message will be known and so, by the pigeonhole principle (remember that $m > r$), there will be messages known to collide. This implies that it is, in some sense, futile to select an output length exceeding $2n$ bits, as $2n$ bits are already enough to accommodate the maximum feasible security[3]. With an output length of $3n$ bits, for example, you'll never get a construction withstanding anything near the optimal value of $q = N^{3/2}$ queries, as no construction can withstand more than $q = N^{1+(\lg k)/n} \ll N^{3/2}$ queries (the "$\ll$" is because we assume that k is a small number).

The *birthday attack* is to compute the permutations necessary to hash $p = q/k$ random messages. By the birthday phenomenon, one expects to see a collision when $p \approx \sqrt{2 \ln 2}\, N^{r/2} \approx 1.18\, N^{r/2}$. For a proper upperbound, note that when $N \geq 2^{16}$, which we will henceforth implicitly assume, the probability of a collision is at least $1/2$ if $p \geq 1.18\, N^{1/2}$ balls are randomly and uniformly thrown into N bins. We record the efficacy of our two attacks in the following proposition.

Proposition 1. *Let $H\colon \{0,1\}^{mn} \to \{0,1\}^{rn}$ be a k-call permutation-based compression function, and let $N = 2^n$. Then with*

$q = kN$ queries an adversary can find a collision with probability 1, and with
$q = 1.18kN^{r/2}$ queries an adversary can find a collision with probability $\geq 1/2$.

[3] This is assuming an information-theoretic adversary, whose only cost is the number of queries made; a "real adversary" may well be hindered by a longer output.

In all theorem statements where, like above, q is an integer but the quantity on the right may be fractional, it is implicit that q is obtained by rounding up the expression on the right. Also, here and subsequently, it is not necessary to restrict m and r to natural number; it is fine to select any rational values m and r such mn and rn are positive integers.

4 The Pigeonhole Attack

We now give a more interesting collision attack on compression functions. It succeeds, always, in about $kN^{1-(m-r)/k}$ queries.

Theorem 1. *Let H: $\{0,1\}^{mn} \to \{0,1\}^{rn}$ be a k-call permutation-based compression function, and let $N = 2^n$. Then with*

$$q = k\,(N^{1-(m-r)/k} + 1) \approx k\,N^{1-(m-r)/k}$$

queries an adversary can find a collision in H with probability 1. □

The concrete consequences of this are interesting. Suppose H is a $2 \xrightarrow{1} 1$ compression function. Then it can be broken in just $q = 2$ queries. So $k = 1$ permutation calls certainly won't do, as shown by Black, Cochran, and Shrimpton [1] in the iterated hash-function setting. In addition, we see that a $2 \xrightarrow{2} 1$ compression function can be broken in about $N^{1/2}$ queries, which is optimal for a hash function of output length n, except that Theorem 1 states the collision can be found with probability 1, whereas an ideal construction would require $2N$ queries for the same result. Quantitative results are tabulated in the top half of Fig. 2.

Proof. Let $p = \lfloor q/k \rfloor$. In brief, the adversary chooses p queries to make to π_1 that enable him to "start" hashing the largest possible number of inputs (each input requires a π_1 query); then the adversary chooses p queries to make to π_2 that will enable him to continue hashing the largest possible number of inputs up to and including the π_2 step; and so on for π_3, …, π_k. If, at the end, the adversary is still able to hash more than N^r inputs, then the adversary wins because some two inputs necessarily collide. The proof simply consists of computing how large p must be for the latter event to happen.

Note first the observation that if B balls are thrown into N bins the $p \leq N$ most occupied bins must contain at least pB/N balls. We will repeatedly use this observation below. Now with the hash function H specified by $f_1, \ldots, f_k$, g, choose a p-element set $X_1 \subseteq \{0,1\}^n$ that has a maximum number of preimages under f_1. By the observation just made, this maximum number of preimages is at least $pN^m/N = pN^{m-1}$ points. The adversary will ask for π_1 at each point $x_1 \in X_1$. The adversary has so far made p queries and there are at least pN^{m-1} points $v \in \{0,1\}^{mn}$ for which the adversary knows how to compute the first permutation in the hash chain. Call this set of points V_1. So $|V_1| \geq pN^{m-1}$ and for each point $v \in V_1$ the adversary knows the corresponding x_1, y_1, and x_2. Next choose p points $X_2 \subseteq \{0,1\}^n$ with a maximum number of $v \in V_1$ that give rise to an $x_2 \in X_2$. Again by the observation that began this paragraph, this set of points V_2 has cardinality $|V_2| \geq p|V_1|/N \geq p^2N^{m-2}$. Continue in this way, selecting a set V_3 where $|V_3| \geq p^3 N^{m-3}$ and making p more queries so that the adversary will know how to compute the beginning computations of a hash value for everything in V_3, knowing everything up to and including the third permutation π_3. Continue until the adversary constructs a set V_k where $|V_k| \geq p^k N^{m-k}$ and the adversary knows how to hash everything in V_k all the way until the end.

If $|V_k| \geq p^k N^{m-k}$ exceeds N^r then, by the pigeonhole principle, there must be two values in V_k that have the same hash, and this hash is known by the adversary we have constructed. Thus the adversary will succeed in finding a collision if $p^k > N^{r-m+k}$, which is to say that it necessarily succeeds if $p > N^{(r-m+k)/k} = N^{1-(m-r)/k}$. So the adversary will find a collision if $\lfloor q/k \rfloor$ exceeds $N^{1-(m-r)/k}$ (hence the chosen value of q). This completes the proof. ∎

5 The Pigeonhole-Birthday Attack

In the proof above we used the fact that a collision is guaranteed as soon as $|V_k| \geq p^k N^{m-k} > N^r$. But it seems unlikely that one would really have to wait so long as that; if the H-outputs computed by the adversary had been random then, by the birthday phenomenon, one would expect to see a collision around the time that $|V_k| = N^{r/2}$, or to be quite exact around the time that $|V_k| = 1.18N^{r/2}$. Let us *assume* that the hash function outputs computed by the adversary in the proof of Theorem 1 behave no worse than random outputs with respect to the appearance of collisions. Call this the *uniformity assumption*. Then solving for the integer p in $p^k N^{m-k} \geq 1.18N^{r/2}$ reveals that we expect to see a collision after $q = kp = k\lceil (1.18)^{1/k} N^{1-(m-0.5r)/k} \rceil \leq k(1 + (1.18)^{1/k} N^{1-(m-0.5r)/k}) \leq k(1 + 1.18\, N^{1-(m-0.5r)/k}) \approx kN^{1-(m-0.5r)/k}$ queries, an improvement from the earlier $q \approx kN^{1-(m-r)/k}$ by a multiplicative factor of $N^{r/2k}$. To summarize:

Theorem 2. *Let $H\colon \{0,1\}^{mn} \to \{0,1\}^{rn}$ be a k-call permutation-based compression function. Let $N = 2^n$. Then, under the uniformity assumption, with*

$$q = k(1 + (1.18)^{1/k} N^{1-(m-0.5r)/k}) \approx k\, N^{1-(m-0.5r)/k}$$

queries an adversary can find a collision with probability $\geq 1/2$. □

The stated bound suffers from a peculiar behavior in the $2 \xrightarrow{k} 1$ case when $k \geq 4$, whence the theorem states that $q \approx kN^{1-3/2k} \geq kN^{5/8}$ queries are sufficient for the attack described, but Proposition 1 ensures that $q = 1.18\, kN^{1/2}$ queries was already enough. The gap may seem more puzzling considering that the pigeonhole-birthday attack *is* a type of birthday attack and, under the uniformity assumption, it cannot do worse than what Proposition 1 guarantees. The problem can be traced to the $p^k N^{m-k}$ lower bound for the number of outputs obtained by the pigeonhole attack, which, in turn, stems from the observation made at the beginning of Theorem 1 that when B balls are thrown into N bins, the $p \leq N$ most occupied bins must contain at least pB/N balls. In fact one can strengthen this observation by noting that the $p \leq N$ most occupied bins must contain at least $\mu_{p,N}(B)$ balls, where $\mu_{p,N}(B)$ is $p\lceil B/N \rceil$ if $p \leq B \bmod n$ or $B \equiv 0 \bmod n$,

atk	adv	asmp	$m \to r$	$\approx$ bound	1	2	3	4	5	6	8
ph	1	no	$2 \to 1$	$kN^{1-1/k}$	2	$2^{65.0}$	$2^{86.9}$	$2^{98.0}$	$2^{104.7}$	$2^{109.3}$	2^{115}
ph	1	no	$3 \to 2$	$kN^{1-1/k}$	2	$2^{65.0}$	$2^{86.9}$	$2^{98.0}$	$2^{104.7}$	$2^{109.3}$	2^{115}
pb	0.5	yes	$2 \to 1$	$1.18kN^{1-3/2k}$	2	$2^{33.1}$	$2^{65.7}$	$2^{66.2}$	$2^{66.6}$	$2^{66.8}$	$2^{67.2}$
pb	0.5	yes	$3 \to 2$	$1.18\,kN^{1-2/k}$	1	3	$2^{44.3}$	$2^{66.1}$	$2^{79.2}$	$2^{88.0}$	$2^{99.0}$

Fig. 2. Attacks on an $m \xrightarrow{k} r$ compression function. Columns are the attack (ph for pigeonhole, pd for pigeonhole-birthday), the adversary's advantage, whether a heuristic assumption is used in the analysis, the compression parameters, the approximate value of q to get this advantage, and numerical values for various values of k, all with $n = 128$.

and $p\lfloor B/N \rfloor + B \bmod N$ otherwise. One thus gets at least $\mu^{(k)}_{p,N}(N^m)$ outputs from the pigeonhole attack (the k-th iterate of the function), better than the approximation $p^k N^{m-k}$. To find the "real" p needed by the attack one can solve for the least integer p such that $\mu^{(k)}_{p,N}(N^m) \geq 1.18N^{r/2}$. As this is somewhat hard to compute, an alternative is to note that, at the end of the pigeonhole-birthday attack, there are at least $p = \lfloor q/k \rfloor$ strings that the adversary knows how to hash, and so $p = 1.18N^{r/2}$ queries are enough (still under the uniformity assumption). We can therefore sharpen the statement of Theorem 2 to select q as the minimum of the current value of q and $1.18\,kN^{r/2} + k \approx 1.18\,kN^{r/2}$, since $p = \lfloor q/k \rfloor > q/k - k$. In Fig. 2 we use this tighter bound to compute the third-row entries.

Interpretation. The bound of the pigeonhole-birthday attack is illustrated numerically in Fig. 2 for $n = 128$ bits. For $2 \to 1$ hashing the analysis indicates that, with $k = 2$ permutations, a collision will be found in around $N^{1/4}$ queries. This is excessively low, making $k = 3$ permutations the best one can hope for in this case. With $k = 3$ permutations the bound jumps to around $N^{1/2}$ queries, which is of course optimal for a hash function producing an n-bit output. This suddenly-optimal behavior is qualitatively different from what happens when the output length is $2n$ bits or more, in which case more permutation calls (potentially) buys more security, but where optimal collision resistance can never reached. For $3 \to 2$ hashing the adversary can break the construction in around $q = N^{1-2/k}$ queries. Since a double-length construction ought to withstand significantly more than $N^{1/2}$ queries (otherwise, it makes more sense to use a single-length construction), the conclusion is that $k = 5$ permutations is the minimum number of calls that makes sense for $3 \to 2$ hashing.

It should be noted that, because of the uniformity assumption, the analysis of Theorem 2 is essentially heuristic. But assumptions analogous to the uniformity assumption are routinely made when analyzing cryptographic attacks, sometimes without even mention that an assumption is being made. And of course one expects that a good hash function *will* have outputs that look uniform on any natural set of inputs produced by an attack.

6 Attacks on Rate-α Constructions

Theorems 1 and 2 can be recast in terms of what they say about a permutation-based hash function with a given rate (as opposed to what they say about a compression function with a given number of blockcipher calls). Let $H : \{0,1\}^* \to \{0,1\}^{rn}$ be a fixed-order hash function based on an n-bit permutation. This means that the algorithm is of the form specified in Fig. 1, except that the input message v may now have any length, and sequences $\pi_1, \pi_2, \pi_3, \ldots$ and $f_1, f_2, f_3, \ldots$ are thought of as infinite, and the number k of permutation invocations is a function $k = k(v)$ of the input v. Then we say that H has *rate* α if α is the largest real number such that hashing a message M requires at most $|M|/\alpha n$ permutation calls. (One could also add in an additive constant δ to account for padding or other extra work done at the end of processing the message.) The *inverse-rate*, $\beta = 1/\alpha$, is the number of permutation calls per n-bits of message processed; hashing M requires at most $\beta|M|/n$ permutation invocations. We now show that the pigeonhole and pigeonhole-birthday attacks imply a tradeoff between the (potential) security of a permutation-based hash function and its rate.

Theorem 3. *Let H: $\{0,1\}^* \to \{0,1\}^{rn}$ be a permutation-based hash function with rate $\alpha = 1/\beta$ and let $N = 2^n$. Then with*

$$q = \lfloor \beta \lceil \ln(2)\alpha nr + \alpha \rceil \rfloor (eN^{1-\alpha} + 1) \approx 1.89\, nrN^{1-\alpha}$$

queries an adversary can find a collision with probability 1. □

Proof. For any $m \geq 1$ we can restrict H to inputs of length mn, whence H becomes a compression function H': $\{0,1\}^{mn} \to \{0,1\}^{rn}$ that makes at most $k = \lfloor \beta m \rfloor$ permutation calls. By Theorem 1, a collision for this compression function can be found with probability 1 in $k(N^{1-(m-r)/k}+1) \leq k(N^{1-\alpha+r/k}+1)$ queries, where again $k = \lfloor \beta m \rfloor$ (the inequality holds because $\alpha \leq m/k$). We set $m = \lceil \ln(2)\alpha nr + \alpha \rceil$ so $k = \lfloor \beta \lceil \ln(2)\alpha nr + \alpha \rceil \rfloor$ (chosen by calculus to minimize $kN^{1-\alpha+r/k}$). Then $k \geq \beta\lceil \ln(2)\alpha nr + \alpha \rceil - 1 \geq \beta(\ln(2)\alpha nr + \alpha) - 1 = \ln(2)nr$ and $N^{r/k} \leq N^{1/\ln(2)n} = e$, so $k(N^{1-\alpha+r/k}+1) \leq \lfloor \beta \lceil \ln(2)\alpha nr + \alpha \rceil \rfloor (eN^{1-\alpha}+1)$, as desired. ∎

One can improve the constant of 1.89 in Theorem 3 by employing the bound of Theorem 2 instead of Theorem 1. Then choosing $m = \lceil ((\ln 2)/2)\, \alpha nr + \alpha \rceil$ yields a final (approximate) bound of $0.94\, nrN^{1-\alpha}$ queries (for generating a collision with probability at least 1/2). Besides halving the probability of success, the price of this change is that one would now need to make the uniformity assumption on the hash function, inherited from Theorem 2, for mn-bit strings.

Ignoring the leading multiplicative and additive factors in Theorem 3 we can summarize the result as saying that any rate-α permutation-based hash function will fail when the number of queries gets to around $q = N^{1-\alpha}$. In Fig. 3 we tabulate this more precisely, indicating the sufficient number of queries to break permutation-based hash functions of various rates.

We comment that, in our result, the number of distinct permutations used by the hash function does not matter, as long as they are consulted in a fixed order.

atk	adv	asmp	bound	restrictions	2	3	4	5	6	8
ph	1	no	$1.89\,nr\,N^{1-\alpha}$	none	$N^{0.57}$	$N^{0.74}$	$N^{0.82}$	$N^{0.87}$	$N^{0.90}$	$N^{0.95}$
pb	0.5	yes	$0.94\,nr\,N^{1-\alpha}$	none	$N^{0.56}$	$N^{0.73}$	$N^{0.81}$	$N^{0.86}$	$N^{0.90}$	$N^{0.95}$
tree	0.5	yes	$2\,\beta\,N^{1-\alpha}$	iterated	$N^{0.52}$	$N^{0.69}$	$N^{0.77}$	$N^{0.83}$	$N^{0.86}$	$N^{0.91}$

Fig. 3. Collision-finding attacks on a permutation-based hash function $H\colon \{0,1\}^* \to \{0,1\}^{rn}$ with rate α. The rows are: the attack (pigeonhole, pigeonhole-birthday, tree); the adversary's advantage; whether a heuristic assumption is used in the analysis; the approximate bound; restrictions on the result; and threshold values q when $n = 128$, $r = 2$, and inverse rates $\beta = 1/\alpha \in \{2, 3, 4, 5, 6, 8\}$.

Potentially, the hash function might never reuse the same permutation twice, but it would still suffer from the same vulnerabilities as long as it consulted its permutations in a prescribed order.

7 Attacking Preimage Resistance

We adopt as a notion of preimage resistance that the adversary is presented a random range point $w \in \{0,1\}^{rn}$ and succeeds if it finds (or simply knows from its query history) a preimage to this point. We first observe that our earlier pigeonhole-attack can be adapted so as to become a preimage-finding attack. We then extend this to give an attack on an arbitrary hash function. As we have chosen our range point at random, neither case requires a heuristic assumption.

Theorem 4. *Let $H\colon \{0,1\}^{mn} \to \{0,1\}^{rn}$ be a k-call permutation-based compression function and let $N = 2^n$. Then with*

$$q = k\,(N^{1-(m-r)/k} + 1) \approx k\,N^{1-(m-r)/k}$$

queries an adversary can invert a random point with probability $\geq 1/2$. □

Proof. The attack proceeds as with the pigeonhole attack, Theorem 1, by greedily constructing a set $V_k \subseteq \{0,1\}^{mn}$ of cardinality at least $p^k N^{m-k}$ for which the adversary knows how to hash everything in V_k. When this set grows to half the size of $\{0,1\}^{rn}$ the adversary will have a 50% chance of inverting a randomly selected point w. So the needed number of queries is the smallest q such that $p^k N^{m-k} \geq 0.5\,N^r$, where $p = \lfloor q/k \rfloor$. Solving, we must ensure that $\lfloor q/k \rfloor \geq (q-k)/k \geq 0.5^{1/k}\,N^{1-(m-r)/k}$. But $0.5^{1/k}\,N^{1-(m-r)/k} \leq N^{1-(m-r)/k}$ so it suffices that $(q-k)/k \geq N^{1-(m-r)/k}$, and the bound follows. ∎

For arbitrary hash functions, as opposed to compression functions, we get the following result to relate preimage resistance to rate.

Theorem 5. *Let $H\colon \{0,1\}^* \to \{0,1\}^{rn}$ be a permutation-based hash function with rate $\alpha = 1/\beta$ and let $N = 2^n$. Then with*

$$q = \lfloor \beta \lceil \ln(2)\alpha nr + \alpha \rceil \rfloor (eN^{1-\alpha} + 1) \approx 1.89\,nrN^{1-\alpha}$$

queries an adversary can invert a random point with probability $1/2$. □

Proof. The proof is exactly the same as for Theorem 3 since the bounds of Theorem 1 and Theorem 4 are the same. ∎

It is interesting that breaking the preimage resistance of a permutation-based hash function is essentially no harder than breaking its collision resistance; our attacks differ in effectiveness only by a factor of 4. In addition, while one may hope to get near-optimal collision resistance with a $2 \xrightarrow{3} 1$ compression function, the preimage resistance will be nowhere near optimal: preimage-resistance will fail by around $N^{2/3}$ queries, whereas one might hope for something that works up to around N queries. But, as with the collision-resistance of double-length constructions, one can hope to push up the preimage resistance to close to N queries by using more and more permutation calls.

8 The Too-Few-Wires Attack

In this section we switch from considering the number of permutations used by a compression function to considering the amount of memory it requires. Mainly we show that a compression function that maps mn bits to rn bits must keep more than mn bits of information in memory at some point during its computation—otherwise it will offer essentially no collision resistance.

Instead of thinking about memory it is useful to think in terms of *wires*. If we imagine that the compression function is built from n-bit wires connecting the permutations and processed at different points by arbitrary functions, our result implies that at least $m + 1$ wires must be used at some point during the computation—one one more wire than there are input wires.

Naturally one needs to define what it means for a compression function to "keep mn bits in memory" during a computation. The model is as follows: we imagine the mn bits to be kept in m "buckets" of n bits each. At any stage, the buckets may either be processed by an arbitrary function $f_i : \{0,1\}^{mn} \to \{0,1\}^{mn}$; or else one of the buckets may be hit with a permutation π_i, replacing the contents of that bucket with the output of the permutation. The buckets are initialized with the input to the compression function, and the computation is terminated by an arbitrary function mapping $\{0,1\}^{mn}$ to $\{0,1\}^{rn}$.

One may assume that no two functions f_i and f_j are ever applied one right after the other (else one could replace them with their composition), and one can assume that permutations are always applied to the first bucket (as the f_i functions can be used to switch bucket contents). Thus if the compression function uses k permutations $(\pi_1, \ldots, \pi_k)$ and we denote by $\bar{\pi}_i$ the map from $\{0,1\}^{mn}$ to $\{0,1\}^{mn}$ that is π_i on the the first bucket and the identity on all others, then the hash of $v \in \{0,1\}^{mn}$ is $f_k(\bar{\pi}_k(f_{k-1}(\bar{\pi}_{k-1}(\ldots f_0(v)\ldots))))$ where f_k: $\{0,1\}^{mn} \to \{0,1\}^{rn}$ and $f_i : \{0,1\}^{mn} \to \{0,1\}^{mn}$ for $i < k$. Figure 4 shows the basic structure, with buckets drawn as wires. The sequence of permutations $(\pi_1, \ldots, \pi_k)$ may be distinct or include repetitions, but we assume that the

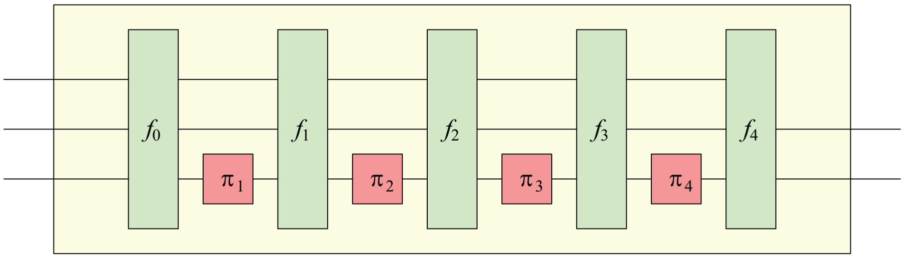


Fig. 4. The structure of a compression function that maps mn bits to rn bits using mn bits of memory where $m = 3$, $r = 2$, and $k = 4$. Each wire represents n bits. Functions f_0, f_1, f_2, f_3, and f_4 are all arbitrary.

permutations are applied in a fixed order, namely that which permutation is applied at a given point does not depend on the contents of the buckets at that point (this restriction can in fact be removed with only a slight increase in the complexity of the attack, so this assumption is mainly made for simplicity). We then have the following:

Theorem 6. *Let H: $\{0,1\}^{mn} \to \{0,1\}^{rn}$ be a permutation-based compression function using k permutation calls and mn bits of memory. Then a collision can be found in $2k$ queries.* □

Proof. With notation as in the paragraph before Theorem 6, let j be the least number such that f_j is not a permutation. Note that j is well-defined since f_k is not a permutation. Fix any two distinct inputs u and v in $\{0,1\}^{mn}$ such that $f_j(u) = f_j(v)$. Because $f_0, \ldots, f_{j-1}$ are permutations we can compute $u' = f_0^{-1}(\bar{\pi}_1^{-1}(f_1^{-1}(\ldots \bar{\pi}_j^{-1}(u) \ldots)))$ and $v' = f_0^{-1}(\bar{\pi}_1^{-1}(f_1^{-1}(\ldots \bar{\pi}_j^{-1}(v) \ldots)))$ with $2j \leq 2k$ permutation calls. Observe that $f_k(\bar{\pi}_k(f_{k-1}(\bar{\pi}_{k-1}(\ldots f_0(u') \ldots))))$ $= f_k(\bar{\pi}_k(f_{k-1}(\bar{\pi}_{k-1}(\ldots f_0(v') \ldots))))$ since $f_j(u) = f_j(v)$ and we are done. ∎

One can generalize this result. Assume that we have at our disposal k ideal primitives $\rho_1, \ldots, \rho_k$, which are functions from $\{0,1\}^{mn}$ to $\{0,1\}^{mn}$ and such that (i) finding a collision for ρ_i costs q_i expected queries to ρ_i, unless ρ_i is a permutation, in which case (ii) finding a preimage for ρ_i costs one query. (An n-bit permutation can be seen as such a primitive, acting only on the first n bits.) A compression function using (ordered) calls $\rho_1, \ldots, \rho_k$ and mn bits of memory can be modeled as above, with mn-bit to mn-bit functions $f_0, \ldots, f_k$ interwoven with $\rho_1, \ldots, \rho_k$. Then one can easily adapt the proof of Theorem 6 to show that the cost of finding a collision for the compression function is at most $\max(q_i) + 2k$, where the max is taken over all i such that ρ_i is not a permutation, and is defined as 0 if all the ρ_i's are permutations. (Proof: take the least j such that either f_j or ρ_j is not a permutation; in the former case let u, v be colliding inputs of f_j, in the latter case let u, v be colliding inputs of ρ_j paid for with q_j

queries; then push back u, v to inputs u', v' for the original function using the fact that all ρ_i's and f_i's for $i < j$ are permutations.)

This observation has some interesting consequences. For example, say that $\rho_1, \ldots, \rho_k$ are random functions from n bits to n bits, so that it costs $2^{n/2}$ queries to find a collision for given ρ_i. Then a compression function from mn bits to rn bits using mn bits of memory, $m > r$, will have collision resistance of at most $2k + 2^{n/2}$, where k is the number of times the random function is called. This is unsatisfactory if $r \geq 2$. It does not matter whether the random functions are distinct or not, nor how many of them are used.

One can also apply the argument to a blockcipher-based construction, say one with n-bit keys and blocks. First define what it means for a blockcipher to "act" on mn bits: one could assume, say, that the first bucket of n bits is used for the blockcipher's key, that the second bucket of n bits is used for the blockcipher's input, and that the blockcipher's output replaces either the first or second bucket. If the blockcipher's output replaces the key, then the blockcipher application is not a permutation and has collision resistance of $2^{n/2}$ (a collision can be obtained by keeping the word constant and tweaking the key); otherwise the blockcipher application constitutes a permutation. Thus, any mn-bit to rn-bit blockcipher-based compression function using only mn-bits of memory in the sense described has collision resistance of $\sim 2^{n/2}$, which is once again unsatisfactory if $r \geq 2$.

As an example of the findings in this section in action, suppose that someone proposes a $3n$-bit to $2n$-bit compression function as shown in Fig. 4, but where we have 10 rounds and each f_i has some combinatorially strong mixing properties. It will not matter that there are a large number of rounds or that the mixing is strong; the scheme will be breakable in a handful of queries. The issue is that the first collision in any of the f_i's can be "pushed back" through the permutations to make two colliding inputs. Then suppose that, to prevent the pushing back, the designer replaces each $x \mapsto \pi_i(x)$ by the feed-forward gadget $x \mapsto x \oplus \pi_i(x)$. Then the number of required wires has gone up by 1, and the attack is blocked. However if we treat the gadget $x \oplus \pi_i(x)$ as a primitive, the number of wires is back down to 3 and the generalized attack shows that a collision can be found in $2^{n/2}$ queries, or the number of queries necessary to find a collision for the gadget $x \oplus \pi_i(x)$. This is insufficient in a scheme that outputs $2n$ bits.

Finally, we comment that it was not important for the attacks of this section that the input length and output length of the compression be multiples of n; all that matters is that the input has at least one more bit than the output.

Acknowledgments

Most of the work on this paper was carried out while the second author was in the Department of Mathematics at UC Davis. Both authors received funding from NSF grant CCR-0208842 and a gift from Intel; many thanks to Intel, particularly Jesse Walker, and to the NSF, for their kind support.

References

1. Black, J., Cochran, M., Shrimpton, T.: On the impossibility of highly-efficient blockcipher-based hash functions. In: Cramer, R.J.F. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 526–541. Springer, Heidelberg (2005)
2. Black, J., Rogaway, P., Shrimpton, T.: Black-box analysis of the block-cipher-based hash-function constructions from PGV. In: Yung, M. (ed.) CRYPTO 2002. LNCS, vol. 2442, pp. 320–335. Springer, Heidelberg (2002)
3. Even, S., Mansour, Y.: A construction of a cipher from a single pseudorandom permutation. In: Matsumoto, T., Imai, H., Rivest, R.L. (eds.) ASIACRYPT 1991. LNCS, vol. 739, pp. 210–224. Springer, Heidelberg (1993)
4. Hirose, S.: How to construct double-block-length hash functions. The second cryptographic hash workshop (sponsored by NIST) (2006)
5. Knudsen, L., Lai, X., Preneel, B.: Attacks on fast double block length hash functions. Journal of Cryptology 11(1), 59–72 (1998)
6. Lucks, S.: A failure-friendly design principle for hash functions. In: Roy, B. (ed.) ASIACRYPT 2005. LNCS, vol. 3788, pp. 474–494. Springer, Heidelberg (2005)
7. Nandi, M.: Towards optimal double-length hash functions. In: Maitra, S., Veni Madhavan, C.E., Venkatesan, R. (eds.) INDOCRYPT 2005. LNCS, vol. 3797, pp. 77–89. Springer, Heidelberg (2005)
8. Preneel, B., Govaerts, R., Vandewalle, J.: On the power of memory in the design of collision resistant hash functions. In: Zheng, Y., Seberry, J. (eds.) AUSCRYPT 2002. LNCS, vol. 718, pp. 105–121. Springer, Heidelberg (1993)
9. Rogaway, P., Steinberger, J.: How to build a permutation-based hash function. Manuscript, available from either author's homepage (2008)
10. Satoh, T., Haga, M., Kurosawa, K.: Towards secure and fast hash functions. IEICE TRANSACTIONS on Fundamentals of Electronics, Communications and Computer Sciences E82-A(1), 55–62 (1999)
11. Shannon, C.: A mathematical theory of communication. Bell System Technical Journal 27, 379–423, 623–656 (1948)

A The Tree Attack

This collision-finding attack is applicable only to an iterated hash function. For that setting and with typical parameters, it does a bit better than the pigeonhole-birthday attack. We describe the attack both for that reason and because it generalizes the interesting attack of Black, Cochran, and Shrimpton [1].

When we say that H is an *iterated* permutation-based hash function we mean that it processes one $sn = (m–r)n$-bit word of message with each iteration, using a compression function H': $\{0,1\}^{mn} \to \{0,1\}^{rn}$. Hash function H is defined by $H(w_1 \cdots w_\ell) = h_\ell$ where $h_i = H'(h_{i-1} \,\|\, w_i)$ and $h_0 \in \{0,1\}^{rn}$, the *initial chaining value*, is a constant. The compression function $H'(h, w)$ is $g(h, w, y_1, \ldots, y_k)$ where $x_i = f_i(h, w, y_1, y_2, \ldots, y_{i-1})$ and $y_i = \pi_i(x_i)$. The construction uses k calls to process sn bits, so its rate is $\alpha = s/k = (m-r)/k$. Natural variants to this model, like letting the compression function H' depend on the position index i, are immaterial in the sequel.

As the name suggests, the tree attack is associated to a certain tree, which we will call the *known-hash tree.* The known-hash tree is constructed deterministically from a set of queries. Before describing anything else, we show how to construct the known-hash tree from a set of queries

The known-hash tree is a subtree of an infinite rooted tree called the *full tree.* The full tree has $k+1$ types of nodes, which we denote type 0, type 1, …, type k. A node of type i has children only of type $i+1$, except for a node of type k, which has children of type 0. The root of the full tree has type 0. Nodes of type $1,\ldots,k$ have outdegree N and nodes of type 0 have outdegree N^s. (As usual, $N = 2^n$.) The outgoing edges from nodes of type $1,\ldots,k$ are labeled with all the values from 0 to $N-1$, whereas the outgoing edges from nodes of type 0 are labeled with all the values from 0 to N^s-1. Every node of type 0 has an associated *value* in $\{0,1\}^{rn}$ defined inductively as follows: the root has value h_0 and a non-root node v of type 0 has value $g(h,w,y_1,\ldots,y_k)$ where h is the value of the first node u of type 0 on the path from v to the root, and where $w,y_1,\ldots,y_k$ are the values on the edges of the path from u to v. Nodes of type $1,\ldots,k$ also have values, defined as follows: the value of a node v of type $i \geq 1$ is $x_i = f_i(h,w,y_1,y_2,\ldots,y_{i-1})$ where h is the value of the first node u of type 0 on the path from v to the root, and where $w,y_1,\ldots,y_{i-1}$ are the values of the edges on the path from u to v.

This completes the description of the full tree. The known-hash tree is a subtree of the full tree. It is defined from a set of queries $\mathcal{Q} = \{(i_1,x_{i_1},y_{i_1}),\ldots,(i_q,x_{i_q},y_{i_q})\}$ made by the adversary, where $\pi_{i_j}(x_{i_j}) = y_{i_j}$ for all $1 \leq j \leq q$. A node v of the full tree is in the known-hash tree if and only if for every node $v_i \neq v$ of type $i \geq 1$ on the path from v to the root the query (i,x_i,y_i) is in $\mathcal{Q}$ where x_i is the value of v_i and where y_i is the value of the outgoing edge of v_i on the path to v. It follows that if v is in the known-hash tree then so are all of its ancestors, so this is defines a valid (but possibly infinite) tree.

If a node v of type 0 is in the known-hash tree then the adversary knows the hash of the word $w_1w_2\cdots w_m$ where $w_1,\ldots,w_m$ are the values of the outgoing edges of the nodes of type 0 on the path from the root to v. This hash is in fact equal to the value of node v. One can also see that every node of type $i \geq 1$ has outdegree ≤ 1 in the known-hash tree, since for every value x_i there is only one y_i such that $\pi_i(x_i) = y_i$. However the outdegree of every node of type 0 is always N^s, since if a node of type 0 is in the known-hash tree then so, by definition, are all of its children. We will call the *reduced outdegree* of a node v of type 0 the number of outgoing edges from v that lie on a path to a node of type 0 further down the tree from v. The *reduced known-hash tree*, or simply *reduced tree*, is the restriction of the known-hash tree to nodes of type 0, where there is an edge from u to v in the reduced tree if and only if u is the first node of type 0 on the path from v to the root in the known-hash tree. Note that the outdegree of a node v in the reduced tree is equal to the reduced outdegree of v in the known-hash tree. One can define a natural bijection from the outgoing edges of v in the reduced tree to those outgoing edges of v in the known-hash tree that lie on a path to some node of type 0 further down. Using this bijection we can label in the natural way the edges of the reduced tree with values from $\{0,1\}^{sn}$. Then every path in the reduced tree corresponds to a word whose hash can be computed by the adversary, with the value of that hash being the value of the terminal node for that path. Thus the reduced tree gives a sort of digest of which hashes the adversary can compute[4] from the queries $\mathcal{Q}$.

For the attack, the adversary will make queries so as to grow the known-hash tree in a greedy fashion. It will make queries to $\pi_1, \ldots, \pi_k$ in cyclical order. When the adversary makes a query to π_i it will choose a value x_i that maximizes the number of terminal nodes of type i in the known-hash tree that have value x_i; that is, the adversary simply chooses the value such that there are a largest possible number of terminal nodes of type i with that value in the known-hash tree (here a *terminal node* is a leaf of the known-hash tree). If there are no terminal nodes of type i, the adversary can make an arbitrary query to π_i. We assume the adversary makes kp queries in all, namely p queries to every permutation. Note that at any given query the known-hash tree could "blow up" and go to infinity; the number of added edges may be much larger than the number of terminal nodes.

This completes the description of the attack. We will now argue that, for q sufficiently large, the adversary has a good chance of obtaining a collision. First note that with kp greedy queries (not the ones we have described above), the pigeonhole argument shows that we can compute the value of the compression function on at least

$$N^{r+s}\left(\frac{p}{N}\right)^k \tag{1}$$

points in the domain $D = \{0,1\}^{r+s}$ of the compression function. This means that the average over the values $h \in \{0,1\}^{rn}$ of the number of points $w \in \{0,1\}^{sn}$ for which we can compute the value of the compression function on input $h \parallel w$ is

$$N^{r+s}\left(\frac{p}{N}\right)^k / N^r = N^s\left(\frac{p}{N}\right)^k . \tag{2}$$

On the other hand, the same average is approximated by the average outdegree of a node in the reduced tree after the adversary has carried out the above tree attack: every node corresponds to a value of h, and every outgoing edge corresponds to a value of w for which the output of the compression function on input $h \parallel w$ is known. The (heuristic) assumption underlying the tree attack is that for moderately large values of p, this outdegree average should approximate the average (2); after all, both the pigeonhole attack and the tree attack choose queries greedily. Then if (2) is moderately large, say equal to 2, we expect the reduced tree to have average outdegree close to 2. But *any* tree with average outdegree exceeding 1 must be infinite, and must also have unbounded width;

[4] The adversary may even know how to compute more hashes than those given from the reduced tree, for example if the function $g(h, w, y_1, \ldots, y_k)$ ignores some of the y_i's, making it not necessary to know their values. However since we are describing an attack and not a proof of security, this is irrelevant.

thus the reduced tree has blown up to infinity and we can find a collision by the pigeonhole principle (and even find a collision at the same level of the tree—meaning a collision of equal-length strings—because the width is unbounded).

To be more concrete, say that we choose $p = q/k$ large enough that

$$N^s \left(\frac{p}{N}\right)^k \geq 2 \tag{3}$$

Then one would expect that with some constant probability close to 1, but say with at least probability 1/2, the tree attack yields a reduced tree of average outdegree exceeding 1. Then the reduced tree has blown up to infinity and we hold a collision. This would give us an attack with probability of success 1/2. The cost of the attack would be $q = kp$ where

$$p = \left\lceil 2^{1/k} N^{1-s/k} \right\rceil \approx 2^{1/k} N^{1-s/k} \ , \tag{4}$$

which is to say $q \approx k\, 2^{1/k}\, N^{1-\alpha} \leq 2k\, N^{1-\alpha}$, because $\alpha = s/k$. This is an improvement on the bound for the pigeonhole-birthday attack since we expect k to be significantly smaller than n.

Theorem 7. *Let H: $\{0,1\}^* \to \{0,1\}^{rn}$ be an iterated permutation-based hash function with rate α, its underlying compression function employing k permutation calls, and let $N = 2^n$. Then, under the heuristic assumptions described above, with*

$$q \approx 2k\, N^{1-\alpha}$$

queries an adversary can find a collision with probability $\geq 1/2$. □

Most iterated hash functions have $s = 1$, in which case $k = k/s = 1/\alpha = \beta$ and the bound of Theorem 7 can be rewritten as $2\beta\, N^{1-\alpha}$; this is the version of the bound used for the numerical examples of Fig. 3. Note that for $\alpha = k = 1$, the case considered by Black *et al.* [1], the tree attack gives a bound of $q = 2$ queries. This may seem seem small, but as Black *et al.* note, any construction in which for any $h, x_1 \in \{0,1\}^n$ there is some $w \in \{0,1\}^n$ such that $x_1 = f_1(h, w)$ can indeed be broken in two queries, using the same argument as for the tree attack (in such a construction, the tree trivially blows up to infinity after just two queries, with uniform reduced outdegree of 2). Moreover, natural constructions will have this feature since it seems undesirable for the function $f_1(h, \cdot)$ to contain collisions (as a function from $\{0,1\}^n$ to $\{0,1\}^n$). However, for constructions that are artificially designed to hold off the attack, the bound $2kN^{1-\alpha}$ may be overly optimistic when it is very small (but in this case one does not much mind being off).