for second preimage resistance or preimage resistance of a hash functions H that require fewer than 2^ℓ evaluations of H (though see Section 5.4.3).

Finding meaningful collisions. The birthday attack just described gives a collision that is not necessarily very useful. But the same idea can be used to find "meaningful" collisions as well. Assume Alice wishes to find two messages x and x' such that $H(x) = H(x')$, and furthermore x should be a letter from her employer explaining why she was fired from work, while x' should be a flattering letter of recommendation. (This might allow Alice to forge an appropriate tag on a letter of recommendation if the hash-and-MAC approach is being used by her employer to authenticate messages.) The observation is that the birthday attack only requires the hash inputs $x_1, \ldots, x_q$ to be distinct; they do not need to be random. Alice can carry out a birthday-type attack by generating $q = \Theta(2^{\ell/2})$ messages of the first type and q messages of the second type, and then looking for collisions between messages of the two types. A small change to the analysis from Appendix A.4 shows that this gives a collision between messages of different types with probability roughly 1/2. A little thought shows that it is easy to write the same message in many different ways. For example, consider the following:

> It is *hard/difficult/challenging/impossible* to *imagine/believe* that we will *find/locate/hire* another *employee/person* having similar *abilities/skills/character* as Alice. She has done a *great/super* job.

Any combination of the italicized words is possible, and expresses the same point. Thus, the sentence can be written in $4 \cdot 2 \cdot 3 \cdot 2 \cdot 3 \cdot 2 = 288$ different ways. This is just one sentence and so it is actually easy to generate a message that can be rewritten in 2^{64} different ways—all that is needed are 64 words with one synonym each. Alice can prepare $2^{\ell/2}$ letters explaining why she was fired and another $2^{\ell/2}$ letters of recommendation; with good probability, a collision between the two types of letters will be found.

5.4.2 Small-Space Birthday Attacks

The birthday attacks described above require a large amount of memory; specifically, they require the attacker to store all $\mathcal{O}(q) = \mathcal{O}(2^{\ell/2})$ values $\{y_i\}$, because the attacker does not know in advance which pair of values will yield a collision. This is a significant drawback because memory is, in general, a scarcer resource than time. It is arguably more difficult to allocate and manage storage for 2^{60} bytes than to execute 2^{60} CPU instructions. Furthermore, one can always let a computation run indefinitely, whereas the memory requirements of an algorithm must be satisfied as soon as that amount of memory is needed.

We show here a better birthday attack with drastically reduced memory requirements. In fact, it has similar time complexity and success probability as before, but uses only *constant* memory. The attack begins by choosing a

random value x_0 and then computing $x_i := H(x_{i-1})$ and $x_{2i} := H(H(x_{2(i-1)}))$ for $i = 1, 2, \ldots$. (Note that $x_i = H^{(i)}(x_0)$ for all i, where $H^{(i)}$ refers to i-fold iteration of H.) In each step the values x_i and x_{2i} are compared; if they are equal then there is a collision somewhere in the sequence $x_0, x_1, \ldots, x_{2i-1}$. The algorithm then finds the least value of j for which $x_j = x_{j+i}$ (note that $j \leq i$ since $j = i$ works), and outputs x_{j-1}, x_{j+i-1} as a collision. This attack, described formally as Algorithm 5.9 and analyzed below, only requires storage of two hash values in each iteration.

ALGORITHM 5.9
A small-space birthday attack

Input: A hash function $H : \{0,1\}^* \to \{0,1\}^\ell$
Output: Distinct x, x' with $H(x) = H(x')$

$x_0 \leftarrow \{0,1\}^{\ell+1}$
$x' := x := x_0$
for $i = 1, 2, \ldots$ **do**:
  $x := H(x)$
  $x' := H(H(x'))$
  // now $x = H^{(i)}(x_0)$ and $x' = H^{(2i)}(x_0)$
  if $x = x'$ **break**
$x' := x$, $x := x_0$
for $j = 1$ to i:
  if $H(x) = H(x')$ **return** x, x' and **halt**
  else $x := H(x)$, $x' := H(x')$
  // now $x = H^{(j)}(x_0)$ and $x' = H^{(i+j)}(x_0)$

How many iterations of the first loop do we expect before $x' = x$? Consider the sequence of values $x_1, x_2, \ldots$, where $x_i = H^{(i)}(x_0)$ as defined before. If we model H as a random function, then each of these values is uniformly and independently distributed in $\{0,1\}^\ell$ until the first repeat occurs. Thus, we expect a repeat to occur with probability $1/2$ in the first $q = \Theta(2^{\ell/2})$ terms of the sequence. We show that when there is a repeat in the first q elements, the algorithm finds a repeat in at most q iterations of the first loop:

CLAIM 5.10 *Let $x_1, \ldots, x_q$ be a sequence of values with $x_m = H(x_{m-1})$. If $x_I = x_J$ with $1 \leq I < J \leq q$, then there is an $i < J$ such that $x_i = x_{2i}$.*

PROOF The sequence $x_I, x_{I+1}, \ldots$ repeats with period $\Delta \stackrel{\text{def}}{=} J - I$. That is, for all $i \geq I$ and $k \geq 0$ it holds that $x_i = x_{i+k\cdot\Delta}$. Let i be the smallest multiple of Δ that is also greater than or equal to I. We have $i < J$ since the sequence of Δ values $I, I+1, \ldots I+(\Delta-1) = J-1$ contains a multiple of Δ. Since $i \geq I$ and $2i - i = i$ is a multiple of Δ, it follows that $x_i = x_{2i}$. ∎

Thus, if there is a repeated value in the sequence $x_1, \ldots, x_q$, then there is some $i < q$ for which $x_i = x_{2i}$. But then in iteration i of our algorithm, we have $x = x'$ and the algorithm breaks out of the first loop. At that point in the algorithm, we know that $x_i = x_{i+i}$. The algorithm then sets $x' := x(= x_i)$ and $x := x_0$, and proceeds to find the *smallest* $j \geq 0$ for which $x_j = x_{j+i}$. (Note $j \neq 0$ because $|x_0| = \ell + 1$.) It outputs x_{j-1}, x_{j+i-1} as a collision.

Finding meaningful collisions. The algorithm just described may not seem amenable to finding meaningful collisions since it has no control over the elements sampled. Nevertheless, we show how finding meaningful collisions is possible. The trick is to find a collision in the right function!

Assume, as before, that Alice wishes to find a collision between messages of two different "types," e.g., a letter explaining why Alice was fired and a flattering letter of recommendation that both hash to the same value. Then, Alice writes each message so that there are $\ell - 1$ interchangeable words in each; i.e., there are $2^{\ell-1}$ messages of each type. Define the one-to-one function $g : \{0,1\}^\ell \to \{0,1\}^*$ such that the ℓth bit of the input selects between messages of type 0 or type 1, and the ith bit (for $1 \leq i \leq \ell - 1$) selects between options for the ith interchangeable word in messages of the appropriate type. For example, consider the sentences:

0: Bob is a *good/hardworking* and *honest/trustworthy worker/employee.*
1: Bob is a *difficult/problematic* and *taxing/irritating worker/employee.*

Define a function g that takes 4-bit inputs, where the last bit determines the type of sentence output, and the initial three bits determine the choice of words in that sentence. For example:

$$g(0000) = \text{Bob is a good and honest worker.}$$
$$g(0001) = \text{Bob is a difficult and taxing worker.}$$
$$g(1010) = \text{Bob is a hardworking and honest employee.}$$
$$g(1011) = \text{Bob is a problematic and taxing employee.}$$

Now define $f : \{0,1\}^\ell \to \{0,1\}^\ell$ by $f(x) \stackrel{\text{def}}{=} H(g(x))$. Alice can find a collision in f using the small-space birthday attack shown earlier. The point here is that any collision x, x' in f yields two messages $g(x), g(x')$ that collide under H. If x, x' is a random collision then we expect that with probability $1/2$ the colliding messages $g(x), g(x')$ will be of different types (since x and x' differ in their final bit with that probability). If the colliding messages are not of different types, the process can be repeated again from scratch.

5.4.3 *Time/Space Tradeoffs for Inverting Functions

In this section we consider the question of preimage resistance, i.e., we are interested in algorithms for the problem of function inversion. Here, an algorithm is given $y = H(x)$ for uniform x, and the goal is to find any x' such