Chapter 5

Hash Functions

A hash function usually means a function that compresses, meaning the output is shorter than the input. Often, such a function takes an input of arbitrary or almost arbitrary length to one whose length is a fixed number, like 160 bits. Hash functions are used in many parts of cryptography, and there are many different types of hash functions, with differing security properties. We will consider them in this chapter.

5.1 The hash function SHA1

The hash function known as SHA1 is a simple but strange function from strings of almost arbitrary length to strings of 160 bits. The function was finalized in 1995, when a FIPS (Federal Information Processing Standard) came out from the US National Institute of Standards that specified SHA1.

Let $\{0,1\}^{<\ell}$ denote the set of all strings of length strictly less than ℓ. The function SHA1: $\{0,1\}^{<2^{64}} \rightarrow \{0,1\}^{160}$ is shown in Fig. 5.1. (Since 2^{64} is a very large length, we think of SHA1 as taking inputs of almost arbitrary length.) It begins by padding the message via the function shapad, and then iterates the *compression function* sha1 to get its output. The operations used in the algorithms of Fig. 5.1 are described in Fig. 5.2. (The first input in the call to SHF1 in code for SHA1 is a 128 bit string written as a sequence of four 32-bit words, each word being consisting of 8 hexadecimal characters. The same convention holds for the initialization of the variable V in the code of SHF1.)

SHA1 is derived from a function called MD4 that was proposed by Ron Rivest in 1990, and the key ideas behind SHA1 are already in MD4. Besides SHA1, another well-known "child" of MD4 is MD5, which was likewise proposed by Rivest. The MD4, MD5, and SHA11 algorithms are all quite similar in structure. The first two produce a 128-bit output, and work by "chaining" a compression function that goes from $512+128$ bits to 128 bits, while SHA1 produces a 160 bit output and works by chaining a compression function from $512+160$ bits to 160 bits.

```
**algorithm** SHA1($M$)    // $|M| < 2^{64}$
    $V \leftarrow$ SHF1( 5A827999 || 6ED9EBA1 || 8F1BBCDC || CA62C1D6 , $M$ )
**return** $V$
```

```
**algorithm** SHF1($K, M$)    // $|K| = 128$ and $|M| < 2^{64}$
    $y \leftarrow \mathsf{shapad}(M)$
    Parse $y$ as $M_1 \| M_2 \| \cdots \| M_n$ where $|M_i| = 512$ $(1 \leq i \leq n)$
    $V \leftarrow$ 67452301 || EFCDAB89 || 98BADCFE || 10325476 || C3D2E1F0
    **for** $i = 1, \ldots, n$ **do**
        $V \leftarrow \mathsf{shf1}(K, M_i \| V)$
**return** $V$
```

```
**algorithm** shapad($M$)    // $|M| < 2^{64}$
    $d \leftarrow (447 - |M|) \bmod 512$
    Let $\ell$ be the 64-bit binary representation of $|M|$
    $y \leftarrow M \| 1 \| 0^d \| \ell$    // $|y|$ is a multiple of 512
**return** $y$
```

```
**algorithm** shf1($K, B \| V$)    // $|K| = 128$, $|B| = 512$ and $|V| = 160$
    Parse $B$ as $W_0 \| W_1 \| \cdots \| W_{15}$ where $|W_i| = 32$ $(0 \leq i \leq 15)$
    Parse $V$ as $V_0 \| V_1 \| \cdots \| V_4$ where $|V_i| = 32$ $(0 \leq i \leq 4)$
    Parse $K$ as $K_0 \| K_1 \| K_2 \| K_3$ where $|K_i| = 32$ $(0 \leq i \leq 3)$
    **for** $t = 16$ to $79$ **do**
        $W_t \leftarrow \mathsf{ROTL}^1(W_{t-3} \oplus W_{t-8} \oplus W_{t-14} \oplus W_{t-16})$
    $A \leftarrow V_0$ ; $B \leftarrow V_1$ ; $C \leftarrow V_2$ ; $D \leftarrow V_3$ ; $E \leftarrow V_4$
    **for** $t = 0$ to $19$ **do**
        $L_t \leftarrow K_0$ ; $L_{t+20} \leftarrow K_1$ ; $L_{t+40} \leftarrow K_2$ ; $L_{t+60} \leftarrow K_3$
    **for** $t = 0$ to $79$ **do**
        **if** $(0 \leq t \leq 19)$ **then** $f \leftarrow (B \wedge C) \vee ((\neg B) \wedge D)$
        **if** $(20 \leq t \leq 39$ OR $60 \leq t \leq 79)$ **then** $f \leftarrow B \oplus C \oplus D$
        **if** $(40 \leq t \leq 59)$ **then** $f \leftarrow (B \wedge C) \vee (B \wedge D) \vee (C \wedge D)$
        $temp \leftarrow \mathsf{ROTL}^5(A) + f + E + W_t + L_t$
        $E \leftarrow D$ ; $D \leftarrow C$ ; $C \leftarrow \mathsf{ROTL}^{30}(B)$ ; $B \leftarrow A$ ; $A \leftarrow temp$
    $V_0 \leftarrow V_0 + A$ ; $V_1 \leftarrow V_1 + B$ ; $V_2 \leftarrow V_2 + C$ ; $V_3 \leftarrow V_3 + D$ ; $V_4 \leftarrow V_4 + E$
    $V \leftarrow V_0 \| V_1 \| V_2 \| V_3 \| V_4$
**return** $V$
```

Figure 5.1: The SHA1 hash function and the underlying SHF1 family.

$X \wedge Y$	bitwise AND of X and Y
$X \vee Y$	bitwise OR of X and Y
$X \oplus Y$	bitwise XOR of X and Y
$\neg X$	bitwise complement of X
$X + Y$	integer sum modulo 2^{32} of X and Y
$\mathsf{ROTL}^l(X)$	circular left shift of bits of X by l positions ($0 \leq l \leq 31$)

Figure 5.2: Operations on 32-bit words used in sha1.

So what is SHA1 supposed to do? First and foremost, it is supposed to be the case that nobody can find distinct strings M and M' such that $\mathsf{SHA1}(M) = \mathsf{SHA1}(M')$. This property is called *collision resistance.*

Stop for a moment and think about the collision-resistance requirement, for it is really quite amazing to think that such a thing could be possible. The function SHA1 maps strings of (almost) any length to strings of 160 bits. So even if you restricted the domain of SHA1 just to “short” strings—let us say strings of length 256 bits—then there must be an *enormous* number of pairs of strings M and M' that hash to the same value. This is just by the pigeonhole principle: if 2^{256} pigeons (the 256-bit messages) roost in 2^{160} holes (the 160-bit hash values) then some two pigeons (two distinct strings) roost in the same hole (have the same hash). Indeed countless pigeons must share the same hole. The difficult is only that nobody has as yet *identified* (meaning, explicitly provided) even two such pigeons (strings).

In trying to define this collision-resistance property of SHA1 we immediately run into “foundational” problems. We would like to say that it is computationally infeasible to output a pair of distinct strings M and M' that collide under SHA1. But in what sense could it be infeasible? There *is* a program—indeed a very short an simple one, having just two “print” statements—whose output specifies a collision. It’s not computationally hard to output a collision; it can’t be. The only difficulty is our *human* problem of not knowing what this program is.

It seems very hard to make a mathematical definition that captures the idea that human beings can’t find collisions in SHA1. In order to reach a mathematically precise definition we are going to have to change the very nature of what we conceive to be a hash function. Namely, rather than it being a single function, it will be a family of functions. This is unfortunate in some ways, because it distances us from concrete hash functions like SHA1. But no alternative is known.

5.2 Collision-resistant hash functions

A hash function for us is a family of functions $H\colon \mathcal{K} \times D \to R$. Here D is the domain of H and R is the range of H. As usual, if $K \in \mathcal{K}$ is a particular key then

Pre-key attack phase	A selects $2 - s$ points
Key selection phase	A key K is selected at random from $\mathcal{K}$
Post-key attack phase	A is given K and returns s points
Winning condition	The 2 points selected by A form a collision for H_K

Figure 5.3: Framework for security notions for collision-resistant hash functions. The three choices of $s \in \{0, 1, 2\}$ give rise to three notions of security.

H_K: $D \to R$ is defined for all $M \in D$ by $H_K(M) = H(K, M)$. This is the instance of H defined by key K.

An example is SHF1: $\{0,1\}^{128} \times \{0,1\}^{<2^{64}} \to \{0,1\}^{160}$, as described in Fig. 5.1. This hash function takes a 128-bit key and an input M of at most 2^{64} bits and returns a 160-bit output. The function SHA1 is an instance of this family, namely the one whose associated key is

$$\texttt{5A827999} \,\|\, \texttt{6ED9EBA1} \,\|\, \texttt{8F1BBCDC} \,\|\, \texttt{CA62C1D6}\ .$$

Let H: $\mathcal{K} \times D \to R$ be a hash function. Here is some notation we use in this chapter. For any key K and $y \in R$ we let

$$H_K^{-1}(y) \ = \ \{\, x \in D \ : \ H_K(x) = y \,\}$$

denote the pre-image set of y under H_K. Let

$$\mathsf{Image}(H_K) \ = \ \{\, H_K(x) \ : \ x \in D \,\}$$

denote the image of H_K.

A *collision* for a function h: $D \to R$ is a pair $x_1, x_2 \in D$ of points such that (1) $H_K(x_1) = H_K(x_2)$ and (2) $x_1 \neq x_2$. The most basic security property of a hash function is collision-resistance, which measures the ability of an adversary to find a collision for an instance of a family H. There are different notions of collision-resistance, varying in restrictions put on the adversary in its quest for a collision.

To introduce the different notions, we imagine a game, parameterized by an integer $s \in \{0, 1, 2\}$, and involving an adversary A. It consists of a *pre-key attack* phase, followed by a *key-selection phase*, followed by a *post-key attack phase*. The adversary is attempting to find a collision for H_K, where key K is selected at random from $\mathcal{K}$ in the key-selection phase. Recall that a collision consists of a pair x_1, x_2 of (distinct) points in D. The adversary is required to specify $2 - s$ points in the pre-key attack phase, before it has any information about the key. (The latter has yet to be selected.) Once the adversary has specified these points and the key has been selected, the adversary is given the key, and will choose the remaining s points as a function of the key, in the post-key attack phase. It wins if the $2 = (2 - s) + s$ points it has selected form a collision for H_K.

Fig. 5.3 summarizes the framework. The three choices of the parameter s give

```
$\mathbf{Exp}_H^{\text{cr2-kk}}(A)$
    $K \xleftarrow{\$} \mathcal{K}$ ; $(x_1, x_2) \xleftarrow{\$} A(K)$
    if ( $H_K(x_1) = H_K(x_2)$ and $x_1 \neq x_2$ and $x_1, x_2 \in D$ )
        then return 1 else return 0
```

```
$\mathbf{Exp}_H^{\text{cr1-kk}}(A)$
    $(x_1, st) \xleftarrow{\$} A()$ ; $K \xleftarrow{\$} \mathcal{K}$ ; $x_2 \xleftarrow{\$} A(K, st)$
    if ( $H_K(x_1) = H_K(x_2)$ and $x_1 \neq x_2$ and $x_1, x_2 \in D$ )
        then return 1 else return 0
```

```
$\mathbf{Exp}_H^{\text{cr0}}(A)$
    $(x_1, x_2) \xleftarrow{\$} A()$ ; $K \xleftarrow{\$} \mathcal{K}$
    if ( $H_K(x_1) = H_K(x_2)$ and $x_1 \neq x_2$ and $x_1, x_2 \in D$ )
        then return 1 else return 0
```

Figure 5.4: Experiments defining security notions for three kinds of collision-resistant hash functions under known-key attack.

rise to three notions of security. The higher the value of s the more power the adversary has, and hence the more stringent is the corresponding notion of security. Fig. 5.4 provides in more detail the experiments underlying the three attacks arising from the above framework. We represent by st information that the adversary wishes to maintain across its attack phases. It will output this information in the pre-key attack phase, and be provided it at the start of the post-key attack phase.

In a variant of this model that we consider in Section 5.8, the adversary is not given the key K in the post-key attack phase, but instead is given an oracle for $H_K(\cdot)$. To disambiguate, we refer to our current notions as capturing collision-resistance under *known-key* attack, and the notions of Section 5.8 as capturing collision-resistance under *hidden-key* attack. The notation in the experiments of Fig. 5.4 and Definition 5.1 reflects this via the use of "kk", except that for CR0, known and hidden key attacks coincide, and hence we just say cr0.

The three types of hash functions we are considering are known by other names in the literature, as indicated in Fig. 5.5.

Definition 5.1 Let $H\colon\ \mathcal{K} \times D \to R$ be a hash function and let A be an algorithm. We let

$$\begin{aligned}
\mathbf{Adv}_H^{\text{cr2-kk}}(A) &= \Pr\left[\mathbf{Exp}_H^{\text{cr2-kk}}(A) = 1\right] \\
\mathbf{Adv}_H^{\text{cr1-kk}}(A) &= \Pr\left[\mathbf{Exp}_H^{\text{cr1-kk}}(A) = 1\right]
\end{aligned}$$

Type	**Name(s) in literature**
CR2-KK	collision-free, collision-resistant, collision-intractable
CR1-KK	universal one-way [8] (aka. target-collision resistant [1])
CR0	universal, almost universal

Figure 5.5: Types of hash functions, with names in our framework and corresponding names found in the literature.

$$\mathbf{Adv}_H^{\mathrm{cr0}}(A) \quad = \quad \Pr\left[\mathbf{Exp}_H^{\mathrm{cr0}}(A) = 1\right] \ . \blacksquare$$

In measuring resource usage of an adversary we use our usual conventions. Although there is formally no definition of a "secure" hash function, we will talk of a hash function being CR2, CR1 or CR0 with the intended meaning that its associated advantage function is small for all adversaries of practical running time.

Note that the running time of the adversary is not really relevant for CR0, because we can always imagine that hardwired into its code is a "best" choice of distinct points x_1, x_2, meaning a choice for which

$$\begin{aligned} &\Pr\left[\, K \xleftarrow{\$} \mathcal{K} \ : \ H_K(x_1) = H_K(x_2) \,\right] \\ &= \ \max_{y_1 \neq y_2} \Pr\left[\, K \xleftarrow{\$} \mathcal{K} \ : \ H_K(y_1) = H_K(y_2) \,\right] \ . \end{aligned}$$

The above value equals $\mathbf{Adv}_H^{\mathrm{cr0}}(A)$ and is the maximum advantage attainable.

Clearly, a CR2 hash function is also CR1 and a CR1 hash function is also CR0. The following states the corresponding relations formally. The proof is trivial and is omitted.

Proposition 5.2 Let $H\colon\ \mathcal{K} \times D \to R$ be a hash function. Then for any adversary A_0 there exists an adversary A_1 having the same running time as A_0 and

$$\mathbf{Adv}_H^{\mathrm{cr0}}(A_0) \ \leq \ \mathbf{Adv}_H^{\mathrm{cr1\text{-}kk}}(A_1) \ .$$

Also for any adversary A_1 there exists an adversary A_2 having the same running time as A_1 and

$$\mathbf{Adv}_H^{\mathrm{cr1\text{-}kk}}(A_1) \ \leq \ \mathbf{Adv}_H^{\mathrm{cr2\text{-}kk}}(A_2) \ . \ \blacksquare$$

We believe that $\mathsf{SHF1}$ is CR2, meaning that there is no practical algorithm A for which $\mathbf{Adv}_H^{\mathrm{cr2\text{-}kk}}(A)$ is appreciably large. This is, however, purely a belief, based on the current inability to find such an algorithm. Perhaps, later, such an algorithm will emerge.

It is useful, for any integer n, to get $\mathsf{SHF1}^n\colon\ \{0,1\}^n \to \{0,1\}^{160}$ denote the restriction of $\mathsf{SHF1}$ to the domain $\{0,1\}^n$. Note that a collision for $\mathsf{SHF1}^n_K$ is also a collision for $\mathsf{SHF1}_K$, and it is often convenient to think of attacking $\mathsf{SHF1}^n$ for some fixed n rather than $\mathsf{SHF1}$ itself.

5.3 Collision-finding attacks

Let us focus on CR2, which is the most important property for the applications we will see later. We consider different types of CR2-type collision-finding attacks on a family $H\colon \mathcal{K} \times D \to R$ where D, R are finite sets. We assume the family performs some reasonable compression, say $|D| \geq 2|R|$. Canonical example families to keep in mind are $H = \mathsf{SHF1}^n$ for $n \geq 161$ and $\mathsf{shf1}$, the compression function of $\mathsf{SHF1}$.

Collision-resistance does not mean it is impossible to find a collision. Analogous to the case of one-wayness, there is an obvious collision-finding strategy. Let us enumerate the elements of D in some way, so that $D = \{D_1, D_2, \ldots, D_d\}$ where $d = |D|$. The following adversary A implements an exhaustive search collision-finding attack:

```
Adversary $A(K)$
    $x_1 \xleftarrow{\$} D$ ; $y \leftarrow H_K(x_1)$
    for $i = 1, \ldots, q$ do
        if ($H_K(D_i) = y$ and $x_1 \neq D_i$) then return $x_1, D_i$
    return FAIL
```

We call q the number of *trials*. Each trial involves one computation of H_K, so the number of trials is a measure of the time taken by the attack. To succeed, the attack requires that $H_K^{-1}(y)$ has size at least two, which happens at least half the time if $|D| \geq 2|R|$. However, we would still expect that it would take about $q = |D|$ trials to find a collision, which is prohibitive, for D is usually large. For example, for $F = \mathsf{shf1}$, the domain has size 2^{672}, far too large. For $\mathsf{SHF1}^n$, we would choose n as small as possible, but need $n \geq 161$ to ensure collisions exist, so the attack uses 2^{161} computations of H_K, which is not practical.

Now here's another idea. We pick points at random, hoping that their image under H_K equals the image under H_K of an initial target point. Call this the random-input collision-finding attack. It is implemented like this:

```
Adversary $A(K)$
    $x_1 \xleftarrow{\$} D$ ; $y \leftarrow H_K(x_1)$
    for $i = 1, \ldots, q$ do
        $x_2 \xleftarrow{\$} D$
        if ($H_K(x_2) = y$ and $x_1 \neq x_2$) then return $x_1, x_2$
    return FAIL
```

A particular trial finds a collision with probability (about) 1 in $|R|$, so we expect to find a collision in about $q = |R|$ trials. This is much better than the $|D|$ trials used by our first attempt. In particular, a collision for $\mathsf{shf1}$ would be found in time

```
for $i = 1, \ldots, q$ do    // $q$ is the number of trials
    $x_i \xleftarrow{\$} D$ ; $y_i \leftarrow H_K(x_i)$
    if (there exists $j < i$ such that $y_i = y_j$ but $x_i \neq x_j$)    // collision found
        then return $x_i, x_j$
    return FAIL    // No collision found
```

Figure 5.6: Birthday attack on a hash function H: $\mathcal{K} \times D \to R$. The attack is successful in finding a collision if it does not return FAIL.

around 2^{160} rather than 2^{672}. But this is still far from practical. Our conclusion is that as long as the range size of the hash function is large enough, this attack is not a threat.

We now consider another strategy, called a *birthday attack*, that turns out to be much better than the above. It is illustrated in Fig. 5.6. It picks at random q points from the domain, and applies H_K to each of them. If it finds two distinct points yielding the same output, it has found a collision for H_K. The question is how large q need be to find a collision. The answer may seem surprising at first. Namely, $q = O(\sqrt{|R|})$ trials suffices.

We will justify this later, but first let us note the impact. Consider $\mathsf{SHA1}^n$ with $n \geq 161$. As we indicated, the random-input collision-finding attack takes about 2^{160} trials to find a collision. The birthday attack on the other hand takes around $\sqrt{2^{160}} = 2^{80}$ trials. This is MUCH less than 2^{160}. Similarly, the birthday attack finds a collision in $\mathsf{shf1}$ in around 2^{80} trials while while random-input collision-finding takes about 2^{160} trials.

To see why the birthday attack performs as well as we claimed, we recall the following game. Suppose we have q balls. View them as numbered, $1, \ldots, q$. We also have N bins, where $N \geq q$. We throw the balls at random into the bins, one by one, beginning with ball 1. At random means that each ball is equally likely to land in any of the N bins, and the probabilities for all the balls are independent. A collision is said to occur if some bin ends up containing at least two balls. We are interested in $C(N, q)$, the probability of a collision. As shown in the Appendix,

$$C(N,q) \;\approx\; \frac{q^2}{2N} \tag{5.1}$$

for $1 \leq q \leq \sqrt{2N}$. Thus $C(N, q) \approx 1$ for $q \approx \sqrt{2N}$.

The relation to birthdays arises from the question of how many people need be in a room before the probability of there being two people with the same birthday is close to one. We imagine each person has a birthday that is a random one of the 365 days in a year. This means we can think of a person as a ball being thrown at random into one of 365 bins, where the i-th bin represents having birthday the i-th day of the year. So we can apply the Proposition from the Appendix with $N = 365$

and q the number of people in the room. The Proposition says that when the room contains $q \approx \sqrt{2 \cdot 365} \approx 27$ people, the probability that there are two people with the same birthday is close to one. This number (27) is quite small and may be hard to believe at first hearing, which is why this is sometimes called the birthday paradox.

To see how this applies to the birthday attack of Fig. 5.6, let us enumerate the points in the range as $R_1, \ldots, R_N$, where $N = |R|$. Each such point defines a bin. We view x_i as a ball, and imagine that it is thrown into bin y_i, where $y_i = H_K(x_i)$. Thus, a collision of balls (two balls in the same bin) occurs precisely when two values x_i, x_j have the same output under H_K. We are interested in the probability that this happens as a function of q. (We ignore the probability that $x_i = x_j$, counting a collision only when $H_K(x_i) = H_K(x_j)$. It can be argued that since D is larger than R, the probability that $x_i = x_j$ is small enough to neglect.)

However, we cannot apply the birthday analysis directly, because the latter assumes that each ball is equally likely to land in each bin. This is not, in general, true for our attack. Let $P(R_j)$ denote the probability that a ball lands in bin R_j, namely the probability that $H_K(x) = R_j$ taken over a random choice of x from D. Then

$$P(y) \;=\; \frac{|H_K^{-1}(R_j)|}{|D|} \;.$$

In order for $P(R_1) = P(R_2) = \cdots = P(R_N)$ to be true, as required to apply the birthday analysis, it must be the case that

$$|H_K^{-1}(R_1)| = |H_K^{-1}(R_2)| = \cdots = |H_K^{-1}(R_N)| \;.$$

A function H_K with this property is called *regular*, and H is called regular if H_K is regular for every K. Our conclusion is that if H is regular, then the probability that the attack succeeds is roughly $C(N, q)$. So the above says that in this case we need about $q \approx \sqrt{2N} = \sqrt{2 \cdot |R|}$ trials to find a collision with probability close to one.

If H is not regular, it turns out the attack succeeds even faster, telling us that we ought to design hash functions to be as "close" to regular as possible [2].

In summary, there is a $2^{l/2}$ or better time attack to find collisions in any hash function outputting l bits. This leads designers to choose l large enough that $2^{l/2}$ is prohibitive. In the case of $\mathsf{SHF1}$ and $\mathsf{shf1}$, the choice is $l = 160$ because 2^{80} is indeed a prohibitive number of trials. These functions cannot thus be considered vulnerable to birthday attack. (Unless they turn out to be extremely non-regular, for which there is no evidence so far.)

Ensuring, by appropriate choice of output length, that a function is not vulnerable to a birthday attack does not, of course, guarantee it is collision resistant. Consider the family $H\colon\ \mathcal{K} \times \{0,1\}^{161} \to \{0,1\}^{160}$ defined as follows. For any K and any x, function $H_K(x)$ returns the first 160 bits of x. The output length is 160, so a birthday attack takes 2^{80} time and is not feasible, but it is still easy to find collisions. Namely, on input K, an adversary can just pick some 160-bit y and output

$y0, y1$. This tells us that to ensure collision-resistance it is not only important to have a long enough output but also design the hash function so that there no clever "shortcuts" to finding a collision, meaning no attacks that exploit some weakness in the structure of the function to quickly find collisions.

We believe that shf1 is well-designed in this regard. Nobody has yet found an adversary that finds a collision in shf1 using less than 2^{80} trials. Even if a somewhat better adversary, say one finding a collision for shf1 in 2^{65} trials, were found, it would not be devastating, since this is still a very large number of trials, and we would still consider shf1 to be collision-resistant.

If we believe shf1 is collision-resistant, Theorem 5.8 tells us that SHF1, as well as $\mathsf{SHF1}_n$, can also be considered collision-resistant, for all n.

5.4 One-wayness of collision-resistant hash functions

Intuitively, a family H is one-way if it is computationally infeasible, given H_K and a range point $y = H_K(x)$, where x was chosen at random from the domain, to find a pre-image of y (whether x or some other) under H_K. Since this definition too has a hidden-key version, we indicate the known-key in the notation below.

Definition 5.3 Let H: $\mathcal{K} \times D \to R$ be a family of functions and let A be an algorithm. We consider the following experiment:

```
$\mathbf{Exp}_H^{\text{ow-kk}}(A)$
    $K \xleftarrow{\$} \mathcal{K}$ ; $x \xleftarrow{\$} D$ ; $y \leftarrow H_K(x)$ ; $x' \xleftarrow{\$} A(K, y)$
    If $(H_K(x') = y$ and $x' \in D)$ then return 1 else return 0
```

We let

$$\mathbf{Adv}_H^{\text{ow-kk}}(A) \quad = \quad \Pr\left[\mathbf{Exp}_H^{\text{ow-kk}}(A) = 1\right] \ . \ \blacksquare$$

We now ask ourselves whether collision-resistance implies one-wayness. It is easy to see, however, that, in the absence of additional assumptions about the hash function than collision-resistance, the answer is "no." For example, let H be a family of functions every instance of which is the identity function. Then H is highly collision-resistant (the advantage of an adversary in finding a collision is zero regardless of its time-complexity since collisions simply don't exist) but is not one-way.

However, we would expect that "genuine" hash functions, meaning ones that perform some non-trivial compression of their data (ie. the size of the range is more than the size of the domain) are one-way. This turns out to be true, but needs to be carefully quantified. To understand the issues, it may help to begin by considering the natural argument one would attempt to use to show that collision-resistance implies one-wayness.

Suppose we have an adversary A that has a significant advantage in attacking the one-wayness of hash function H. We could try to use A to find a collision via the following strategy. In the pre-key phase (we consider a type-1 attack) we pick and return a random point x_1 from D. In the post-key phase, having received the key K, we compute $y = H_K(x_1)$ and give K, y to A. The latter returns some x_2, and, if it was successful, we know that $H_K(x_2) = y$. So $H_K(x_2) = H_K(x_1)$ and we have a collision.

Not quite. The catch is that we only have a collision if $x_2 \neq x_1$. The probability that this happens turns out to depend on the quantity:

$$\mathbf{PreIm}_H(1) \;=\; \Pr\left[\, K \xleftarrow{\$} \mathcal{K} \,;\, x \xleftarrow{\$} D \,;\, y \leftarrow H_K(x) \;:\; |H_K^{-1}(y)| = 1 \,\right] .$$

This is the probability that the size of the pre-image set of y is exactly 1, taken over y generated as shown. The following Proposition says that a collision-resistant function H is one-way as long as $\mathbf{PreIm}_H(1)$ is small.

Proposition 5.4 Let H: $\mathcal{K} \times D \to R$ be a hash function. Then for any A there exists a B such that

$$\mathbf{Adv}_H^{\text{ow-kk}}(A) \;\leq\; 2 \cdot \mathbf{Adv}_H^{\text{cr1-kk}}(B) + \mathbf{PreIm}_H(1) .$$

Furthermore the running time of B is that of A plus the time to sample a domain point and compute H once. ▌

The result is about the CR1 type of collision-resistance. However Proposition 5.2 implies that the same is true for CR2.

A general and widely-applicable corollary of the above Proposition is that collision-resistance implies one-wayness as long as the domain of the hash function is significantly larger than its range. The following quantifies this.

Corollary 5.5 Let H: $\mathcal{K} \times D \to R$ be a hash function. Then for any A there exists a B such that

$$\mathbf{Adv}_H^{\text{ow-kk}}(A) \;\leq\; 2 \cdot \mathbf{Adv}_H^{\text{cr1-kk}}(B) + \frac{|R|}{|D|} .$$

Furthermore the running time of B is that of A plus the time to sample a domain point and compute H once. ▌

Proof of Corollary 5.5: For any key K, the number of points in the range of H_K that have exactly one pre-image certainly cannot exceed $|R|$. This implies that

$$\mathbf{PreIm}_H(1) \;\leq\; \frac{|R|}{|D|} .$$

The corollary follows from Proposition 5.4. ▌

Corollary 5.5 says that if H is collision-resistant, and performs enough compression that $|R|$ is much smaller than $|D|$, then it is also one-way. Why? Let A be a practical adversary that attacks the one-wayness of H. Then B is also practical, and since H is collision-resistant we know $\mathbf{Adv}_H^{\text{cr1-kk}}(B)$ is low. Equation (5.2) then tells us that as long as $|R|/|D|$ is small, $\mathbf{Adv}_H^{\text{ow-kk}}(A)$ is low, meaning H is one-way.

As an example, let H be the compression function $\mathsf{shf1}$. In that case $R = \{0,1\}^{160}$ and $D = \{0,1\}^{672}$ so $|R|/|D| = 2^{-512}$, which is tiny. We believe $\mathsf{shf1}$ is collision-resistant, and the above thus says it is also one-way.

There are some natural hash functions, however, for which Corollary 5.5 does not apply. Consider a hash function H every instance of which is two-to-one. The ratio of range size to domain size is $1/2$, so the right hand side of the equation of Corollary 5.5 is 1, meaning the bound is vacuous. However, such a function is a special case of the one considered in the following Proposition.

Corollary 5.6 Suppose $1 \leq r < d$ and let $H \colon \mathcal{K} \times \{0,1\}^d \to \{0,1\}^r$ be a hash function which is regular, meaning $|H_K^{-1}(y)| = 2^{d-r}$ for every $y \in \{0,1\}^r$ and every $K \in \mathcal{K}$. Then for any A there exists a B such that

$$\mathbf{Adv}_H^{\text{ow-kk}}(A) \quad \leq \quad 2 \cdot \mathbf{Adv}_H^{\text{cr1-kk}}(B) \; .$$

Furthermore the running time of B is that of A plus the time to sample a domain point and compute H once. ▌

Proof of Corollary 5.6: The assumption $d > r$ implies that $\mathbf{PreIm}_H(1) = 0$. Now apply Proposition 5.4. ▌

We now turn to the proof of Proposition 5.4.

Proof of Proposition 5.4: Here's how B works:

```
Pre-key phase

Adversary $B()$
    $x_1 \xleftarrow{\$} D$ ; $st \leftarrow x_1$
    return $(x_1, st)$
```

```
Post-key phase

Adversary $B(K, st)$
    Retrieve $x_1$ from $st$
    $y \leftarrow H_K(x_1)$ ; $x_2 \xleftarrow{\$} B(K, y)$
    return $x_2$
```

Let $\Pr[\cdot]$ denote the probability of event "$\cdot$" in experiment $\mathbf{Exp}_H^{\text{cr1-kk}}(B)$. For any $K \in \mathcal{K}$ let

$$S_K \;=\; \{\, x \in D \;:\; |H_K^{-1}(H_K(x))| = 1 \,\} \; .$$

$$\begin{aligned}
&\mathbf{Adv}_H^{\text{cr1-kk}}(B) && (5.2)\\
&\quad = \;\; \Pr\left[H_K(x_2) = y \wedge x_1 \neq x_2\right] && (5.3)\\
&\quad \geq \;\; \Pr\left[H_K(x_2) = y \wedge x_1 \neq x_2 \wedge x_1 \notin S_K\right] && (5.4)\\
&\quad = \;\; \Pr\left[x_1 \neq x_2 \mid H_K(x_2) = y \wedge x_1 \notin S_K\right] \cdot \Pr\left[H_K(x_2) = y \wedge x_1 \notin S_K\right] && (5.5)
\end{aligned}$$

$$\geq \frac{1}{2} \cdot \Pr\left[H_K(x_2) = y \wedge x_1 \notin S_K\right] \tag{5.6}$$

$$\geq \frac{1}{2} \cdot \left(\Pr\left[H_K(x_2) = y\right] - \Pr\left[x_1 \in S_K\right]\right) \tag{5.7}$$

$$= \frac{1}{2} \cdot \left(\mathbf{Adv}^{\text{ow-kk}}_{H}(A) - \mathbf{PreIm}_H(1)\right) . \tag{5.8}$$

Re-arranging terms yields Equation (5.2). Let us now justify the steps above. Equation (5.3) is by definition of $\mathbf{Adv}^{\text{cr1-kk}}_{H}(B)$ and B. Equation (5.4) is true because $\Pr[E] \geq \Pr[E \wedge F]$ for any events E, F. Equation (5.5) uses the standard formula $\Pr[E \wedge F] = Pr[E|F] \cdot \Pr[F]$. Equation (5.6) is justified as follows. Adversary A has no information about x_1 other than that it is a random point in the set $H_K^{-1}(y)$. However if $x_1 \notin S_K$ then $|H_K^{-1}(y)| \geq 2$. So the probability that $x_2 \neq x_1$ is at least 1/2 in this case. Equation (5.7) applies another standard probabilistic inequality, namely that $\Pr[E \wedge \overline{F}] \geq \Pr[E] - \Pr[F]$. Equation (5.8) uses the definitions of the quantities involved. ▌

5.5 The MD transform

We saw above that SHF1 worked by iterating applications of its compression function shf1. The latter, under any key, compresses 672 bits to 160 bits. SHF1 works by compressing its input 512 bits at a time using shf1.

The iteration method has been chosen carefully. It turns out that if shf1 is collision-resistant, then SHF1 is guaranteed to be collision-resistant. In other words, the harder task of designing a collision-resistant hash function taking long and variable-length inputs has been reduced to the easier task of designing a collision-resistant compression function that only takes inputs of some fixed length.

This has clear benefits. We need no longer seek attacks on SHF1. To validate it, and be assured it is collision-resistant, we need only concentrate on validating shf1 and showing the latter is collision-resistant.

This is one case of an important hash-function design principle called the MD paradigm [10, 3]. This paradigm shows how to transform a compression function into a hash function in such a way that collision-resistance of the former implies collision-resistance of the latter. We are now going to take a closer look at this paradigm.

Let b be an integer parameter called the block length, and v another integer parameter called the chaining-variable length. Let h: $\mathcal{K} \times \{0,1\}^{b+v} \rightarrow \{0,1\}^v$ be a family of functions that we call the compression function. We assume it is collision-resistant.

Let B denote the set of all strings whose length is a positive multiple of b bits, and let D be some subset of $\{0,1\}^{<2^b}$.

Definition 5.7 A function pad: $D \rightarrow B$ is called a *MD-compliant padding function* if it has the following properties for all $M, M_1, M_2 \in D$:

```
$H(K, M)$
    $y \leftarrow \mathsf{pad}(M)$
    Parse $y$ as $M_1 \,\|\, M_2 \,\|\, \cdots \,\|\, M_n$ where $|M_i| = b$ $(1 \le i \le n)$
    $V \leftarrow \mathtt{IV}$
    for $i = 1, \ldots, n$ do
        $V \leftarrow h(K, M_i \,\|\, V)$
Return $V$
```

```
Adversary $A_h(K)$
    Run $A_H(K)$ to get its output $(x_1, x_2)$
    $y_1 \leftarrow \mathsf{pad}(x_1)$ ; $y_2 \leftarrow \mathsf{pad}(x_2)$
    Parse $y_1$ as $M_{1,1} \,\|\, M_{1,2} \,\|\, \cdots \,\|\, M_{1,n[1]}$ where $|M_{1,i}| = b$ $(1 \le i \le n[1])$
    Parse $y_2$ as $M_{2,1} \,\|\, M_{2,2} \,\|\, \cdots \,\|\, M_{2,n[2]}$ where $|M_{2,i}| = b$ $(1 \le i \le n[2])$
    $V_{1,0} \leftarrow \mathtt{IV}$ ; $V_{2,0} \leftarrow \mathtt{IV}$
    for $i = 1, \ldots, n[1]$ do $V_{1,i} \leftarrow h(K, M_{1,i} \,\|\, V_{1,i-1})$
    for $i = 1, \ldots, n[2]$ do $V_{2,i} \leftarrow h(K, M_{2,i} \,\|\, V_{2,i-1})$
    if $(V_{1,n[1]} \ne V_{2,n[2]}$ OR $x_1 = x_2)$ return FAIL
    if $|x_1| \ne |x_2|$ then return $(M_{1,n[1]} \,\|\, V_{1,n[1]-1}, M_{2,n[2]} \,\|\, V_{2,n[2]-1})$
    $n \leftarrow n[1]$    // $n = n[1] = n[2]$ since $|x_1| = |x_2|$
    for $i = n$ downto 1 do
        if $M_{1,i} \,\|\, V_{1,i-1} \ne M_{2,i} \,\|\, V_{2,i-1}$ then return $(M_{1,i} \,\|\, V_{1,i-1}, M_{2,i} \,\|\, V_{2,i-1})$
```

Figure 5.7: Hash function H defined from compression function h via the MD paradigm, and adversary A_h for the proof of Theorem 5.8.

(1) M is a prefix of $\mathsf{pad}(M)$

(2) If $|M_1| = |M_2|$ then $|\mathsf{pad}(M_1)| = |\mathsf{pad}(M_2)|$

(3) If $M_1 \ne M_2$ then the last block of $\mathsf{pad}(M_1)$ is different from the last block of $\mathsf{pad}(M_2)$. ▌

A block, above, consists of b bits. Remember that the output of pad is in B, meaning is a sequence of b-bit blocks. Condition (3) of the definition is saying that if two messages are different then, when we apply pad to them, we end up with strings that differ in their final blocks.

An example of a MD-compliant padding function is shapad. However, there are other examples as well.

Now let IV be a v-bit value called the initial vector. We build a family $H\colon \mathcal{K} \times D \to \{0,1\}^v$ from h and pad as illustrated in Fig. 5.7. Notice that SHF1 is such a family, built from $h = \mathsf{shf1}$ and $\mathsf{pad} = \mathsf{shapad}$. The main fact about this method is the following.

Theorem 5.8 Let h: $\mathcal{K} \times \{0,1\}^{b+v} \rightarrow \{0,1\}^v$ be a family of functions and let H: $\mathcal{K} \times D \rightarrow \{0,1\}^v$ be built from h as described above. Suppose we are given an adversary A_H that attempts to find collisions in H. Then we can construct an adversary A_h that attempts to find collisions in h, and

$$\mathbf{Adv}_H^{\text{cr2-kk}}(A_H) \;\leq\; \mathbf{Adv}_h^{\text{cr2-kk}}(A_h)\;. \tag{5.9}$$

Furthermore, the running time of A_h is that of A_H plus the time to perform $(|\mathsf{pad}(x_1)| + |\mathsf{pad}(x_2)|)/b$ computations of h where (x_1, x_2) is the collision output by A_H. ▌

This theorem says that if h is collision-resistant then so is H. Why? Let A_H be a practical adversary attacking H. Then A_h is also practical, because its running time is that of A_H plus the time to do some extra computations of h. But since h is collision-resistant we know that $\mathbf{Adv}_h^{\text{cr2-kk}}(A_h)$ is low. Equation (5.9) then tells us that $\mathbf{Adv}_H^{\text{cr2-kk}}(A_H)$ is low, meaning H is collision-resistant as well.

Proof of Theorem 5.8: Adversary A_h, taking input a key $K \in \mathcal{K}$, is depicted in Fig. 5.7. It runs A_H on input K to get a pair (x_1, x_2) of messages in D. We claim that if x_1, x_2 is a collision for H_K then A_h will return a collision for h_K.

Adversary A_h computes $V_{1,n[1]} = H_K(x_1)$ and $V_{2,n[2]} = H_K(x_2)$. If x_1, x_2 is a collision for H_K then we know that $V_{1,n[1]} = V_{2,n[2]}$. Let us assume this. Now, let us look at the inputs to the application of h_K that yielded these outputs. If these inputs are different, they form a collision for h_K.

The inputs in question are $M_{1,n[1]} \parallel V_{1,n[1]-1}$ and $M_{2,n[2]} \parallel V_{2,n[2]-1}$. We now consider two cases. The first case is that x_1, x_2 have different lengths. Item (3) of Definition 5.7 tells us that $M_{1,n[1]} \neq M_{2,n[2]}$. This means that $M_{1,n[1]} \parallel V_{1,n[1]-1} \neq M_{2,n[2]} \parallel V_{2,n[2]-1}$, and thus these two points form a collision for h_K that can be output by A_h.

The second case is that x_1, x_2 have the same length. Item (2) of Definition 5.7 tells us that y_1, y_2 have the same length as well. We know this length is a positive multiple of b since the range of pad is the set B, so we let n be the number of b-bit blocks that comprise y_1 and y_2. Let V_n denote the value $V_{1,n}$, which by assumption equals $V_{2,n}$. We compare the inputs $M_{1,n} \parallel V_{1,n-1}$ and $M_{2,n} \parallel V_{2,n-1}$ that under h_K yielded V_n. If they are different, they form a collision for h_K and can be returned by A_h. If, however, they are the same, then we know that $V_{1,n-1} = V_{2,n-1}$. Denoting this value by V_{n-1}, we now consider the inputs $M_{1,n-1} \parallel V_{1,n-2}$ and $M_{2,n-1} \parallel V_{2,n-2}$ that under h_K yield V_{n-1}. The argument repeats itself: if these inputs are different we have a collision for h_K, else we can step back one more time.

Can we get stuck, continually stepping back and not finding our collision? No, because $y_1 \neq y_2$. Why is the latter true? We know that $x_1 \neq x_2$. But item (1) of Definition 5.7 says that x_1 is a prefix of y_1 and x_2 is a prefix of y_2. So $y_1 \neq y_2$.

We have argued that on any input K, adversary A_h finds a collision in h_K exactly when A_H finds a collision in H_K. This justifies Equation (5.9). We now justify the claim about the running time of A_h. The main component of the running time of A_h is the time to run A_H. In addition, it performs a number of computations of h equal to the number of blocks in y_1 plus the number of blocks in y_2. There is some more overhead, but small enough to neglect. ■

5.6 Polynomial evaluation is an almost-universal hash function

5.7 The CBC MAC is an almost-universal hash function

We show that if $M, M' \in (\{0,1\}^n)^+$ are distinct then $\Pr[\rho \xleftarrow{\$} \mathsf{Func}(n) :\ \mathrm{CBC}_\rho(M) = \mathrm{CBC}_\rho(M')]$ is small. By "small" we mean a slowly growing function of $m = \|M\|_n$ and $m' = \|M'\|_n$. We use another game-playing argument.

Lemma 5.9 [CBC Collision Bound] Fix $n, m, m' \geq 1$ and let $M \in (\{0,1\}^n)^m$ and $M' \in (\{0,1\}^n)^{m'}$ be distinct strings. Then

$$\mathbf{Coll}^{m,\, m'}_{\mathrm{CBC}[\mathsf{Func}(n)]} \ \leq\ \frac{mm'}{2^n} + \frac{\max\{m, m'\}}{2^n}$$

Proof: If the lengths of M and M' differ then, without loss of generality, let M name the shorter of the two strings and let M' name the longer. Then write $M = M_1 \cdots M_m$ and $M' = M'_1 \cdots M'_{m'}$ where each M_i and each M'_j is n-bits long and $m' \geq m$. Write M as $M = N \,\|\, I$ and write M' as $M' = N \,\|\, I'$ where N is the longest common prefix of M and M' whose length is divisible by n. Let $k = \|N\|_n$. Note that $k = m$ if M is a prefix of M' and otherwise k is the largest nonnegative integer such that $M_1 \cdots M_k = M'_1 \cdots M'_k$ but $M_{k+1} \neq M'_{k+1}$. By definition, if $M_1 \neq M'_1$ then $k = 0$. Further note that $k < m'$.

Consider game C1, as defined in Fig. 5.8. This game realizes one way to compute $Y_m = \mathrm{CBC}_\rho(M)$ and $Y'_{m'} = \mathrm{CBC}_\rho(M')$ for a random $\rho \in \mathsf{Func}(n)$, and so we would like to bound the probability, in game C1, that $Y_m = Y'_{m'}$. Also depicted in Fig. 5.8 is game C2, obtained by eliminating the shaded statements.

Before taking up the analysis of the games in earnest, we give a bit of intuition about what they aim to capture. We are interested in the probability that $Y_m = Y'_{m'}$. This can happen due either to an *internal collision* between X_m and $X'_{m'}$, where Y_m was produced as $\rho(X_m)$ and $Y'_{m'}$ was produced as $\rho(X'_{m'})$, or Y_m might equal $Y'_{m'}$ even in the absence of this collsion between X_m and $X'_{m'}$. Intuitively, the latter seems unlikely, and it is. The former is unlikely too, but to prove this we generalize and consider a broader set of internal collisions than just X_m coinciding with $X'_{m'}$. That is, consider the $X_1, \ldots, X_m$ values that get fed to ρ as we process $M = M_1 \cdots M_m$, and consider the $X'_1, \ldots, X'_{m'}$ values that get fed to ρ as we process $M' = M'_1 \cdots M'_{m'}$.

```
$bad \leftarrow \mathsf{false}$
for $X \in \{0,1\}^n$ do $\eta(X) \leftarrow \iota(X) \leftarrow \iota'(X) \leftarrow \mathtt{undef}$

$Y_0 \leftarrow 0^n$
for $i \leftarrow 1$ to $k$ do
    $Y_i \xleftarrow{\$} \{0,1\}^n$
    $X_i \leftarrow Y_{i-1} \oplus M_i$
    if $X_i \in \mathrm{Domain}(\eta)$ then $Y_i \leftarrow \eta(X_i)$, $bad \leftarrow \mathsf{true}$ else $\eta(X_i) \leftarrow Y_i$

for $i \leftarrow k+1$ to $m$ do
    $Y_i \xleftarrow{\$} \{0,1\}^n$
    $X_i \leftarrow Y_{i-1} \oplus M_i$
    if $X_i \in \mathrm{Domain}(\iota)$ then $Y_i \leftarrow \iota(X_i)$ else $\iota(X_i) \leftarrow Y_i$
    if $X_i \in \mathrm{Domain}(\eta)$ then $Y_i \leftarrow \eta(X_i)$, $bad \leftarrow \mathsf{true}$

$Y'_k \leftarrow Y_k$
for $j \leftarrow k+1$ to $m'$ do
    $Y'_j \xleftarrow{\$} \{0,1\}^n$
    if $Y'_j = Y_m$ then $bad \leftarrow \mathsf{true}$
    $X'_j \leftarrow Y'_{j-1} \oplus M'_j$
    if $X'_j \in \mathrm{Domain}(\iota')$ then $Y_j \leftarrow \iota'(X'_j)$ else $\iota'(X'_j) \leftarrow Y'_j$
    if $X'_j \in \mathrm{Domain}(\iota)$ then $Y'_j \leftarrow \iota(X'_j)$, $bad \leftarrow \mathsf{true}$

    if $X'_j \in \mathrm{Domain}(\eta)$ then $Y'_j \leftarrow \eta(X'_j)$, $bad \leftarrow \mathsf{true}$
```

Figure 5.8: Game C1, as written, and game C2, after eliminating the shaded statements. Game C1 provides a way to compute the CBC MAC of distinct messages $M = M_1 \cdots M_m$ and $M' = M'_1 \cdots M'_{m'}$ that are identical up to block k. The computed MACs are Y_m and $Y'_{m'}$.

Accounting for the fact that $X_i = X'_i$ for all $i \in [1 \mathinner{..} k]$ (that is, the X_i-values that occur as we process the common prefix N of M and M'), we don't really expect to see collisions among $X_1, \ldots, X_m, X'_{i+1}, \ldots, X'_{m'}$. To get a better bound, we will "give up" in the analysis for some of these possibilities, but not all. Specifically, the internal collisions that we focus on are (a) collisions that occur while we process N (meaning collisions among any of $X_1, \ldots, X_k$); (b) collisions that occur between N and I (those between one of $X_1, \ldots, X_k$ and one of $X_{k+1}, \ldots, X_m$); (c) collisions that occur between N and I' (one of $X_1, \ldots, X_k$ coincides with one of $X'_{k+1}, \ldots, X'_{m'}$); and (d) collisions that occur between I' and I (one of $X_{k+1}, \ldots, X_m$ coincides with one of $X'_{k+1}, \ldots, X'_{m'}$). There is nothing "magical" about giving up exactly on these internal collisions—it is simply that the more internal collisions one gives up on the worse the bound but the easier the analysis. The proof we give formalizes the intuition of this paragraph and does all the necessary accounting.

Now look at game C1. It works not by growing a single random function ρ but by growing *three* random functions: η, ι, and ι'. The function η keeps track of the association of points that arise during the processing of N, the function ι keeps track of the association of points that arise during the processing of I, and the function ι' keeps track of the association of points that arise during the processing of I'. If a value X should get placed in the domain of two or more of these three functions then we regard the corresponding range value as that specified first by η if such a value has been specified, and as the value secondarily by ι otherwise. Game C1 can thus be seen to provide a perfect simulation of the CBC algorithm over a random function $\rho \in \mathsf{Func}(n)$, and we seek to bound the probability, in game C1, that $Y_m = Y'_{m'}$.

We first claim that, in game C1, any time that $Y_m = Y'_{m'}$ it is also the case that flag *bad* gets set to true. Focus on line 42. Since $m' \geq k+1$ we know that the loop covering lines 40–46 will be executed at least once and $Y'_{m'}$ will take its value as a result of these statements. When a preliminary $Y'_{m'}$ value gets chosen at line 41 for iteration $j = m'$ we see that if $Y'_{m'} = Y_m$ then line 42 will set the flag *bad*. But observe that the initially chosen $Y'_{m'}$ value is not necessarily the final $Y'_{m'}$ value returned by game C1—the $Y'_{m'}$ value chosen at line 42 could get overwritten at any of lines line 44, 45, or 46. If the value gets overwritten at either of line 45 or line 46 then *bad* will get set to true. If $Y'_{m'}$ gets overwritten by Y_m at line 44 then Y_m had earlier been placed in the range of ι' and it could only have gotten there (since ι' grows only by the **else** -clause of line 44) by $Y'_{m'}$ being equal to Y'_j for an already selected $j \in [k+1 \mathinner{..} m'-1]$. In that case the flag *bad* would have already been set to true by an earlier execution of line 42: when the Y'_j value was randomly selected at line 41 and found to coincide with Y_m the flag *bad* is set. We thus have that every execution of game C1 that results in $Y_m = Y'_{m'}$ also results in *bad* being true.

Let $\Pr_1[\cdot]$ denote the probability of an event in game C1 and let $\Pr_2[\cdot]$ denote the probability of an event in game C2. Let B be the event (whether in game C1 or game C2) that the flag *bad* gets set to true. By the contents of the last paragraph, $\Pr_\rho[\mathrm{CBC}_\rho(M) = \mathrm{CBC}_\rho(M')] \leq \Pr_1[\mathsf{B}]$. Also note that games C1 and C2 are identical until the flag *bad* gets set to true (meaning that any execution in which a highlighted statement is executed it is also the case that *bad* is set to true in that execution) and so, in particular, $\Pr_1[\mathsf{B}] = \Pr_2[\mathsf{B}]$. We thus have that $\Pr_\rho[\mathrm{CBC}_\rho(M) = \mathrm{CBC}_\rho(M')] \leq \Pr_2[\mathsf{B}]$. We now proceed to bound $\Pr_2[\mathsf{B}]$, showing that $\Pr_2[\mathsf{B}] \leq mm'/2^n + \max\{m, m'\}/2^n$. This is done by summing the probabilities that *bad* gets set to true at lines 14, 24, 42, 45, and 46.

The probability that *bad* gets set at **line 14** of game C2 is at most $0.5\, k(k-1)/2^n$. This is because every point placed into the domain of η is of the form $X_i = Y_{i-1} \oplus M_i$ where each Y_{i-1} is randomly selected from $\{0,1\}^n$ (at line 12) with the single exception of Y_0, which is a constant. So for any i and j with $1 \leq i < j \leq k$ we have that $\Pr[Y_{i-1} \oplus M_i = Y_{j-1} \oplus M_j] = 2^{-n}$ and there are $0.5\, k(k-1)/2^n$ such

(i, j) pairs possible.

The probability that *bad* gets set at **line 24** of game C2 is at most $k(m-k)/2^n$. Each point X_i whose presence in the domain of η we test is of the form $X_i = Y_{i-1} \oplus M_i$, as defined at line 22. Each of these Y_{i-1} values with the exception of Y_k is uniformly selected at line 21, independent of the now-determined domain of η. As for $X_{k+1} = Y_k \oplus M_{k+1}$, the value Y_k was just selected uniformly at random (at the last execution of line 12) and is independent of the domain of η. Thus each time line 24 executes there is at most a $k/2^n$ chance that *bad* will get set, and the line is executed $m-k$ times.

The probability that *bad* gets set at **line 42** of game C2 is at most $(m'-k)/2^n$. This is because the line is executed $m'-k$ times and each time the chance that *bad* gets set is $1/2^n$ since Y'_j was just selected at random from $\{0,1\}^n$.

The probability that *bad* gets set at **line 45** of game C2 is at most $(m-k)(m'-k)$. Each point X'_j whose presence in the domain of ι is being tested is of the form $X'_j = Y'_{j-1} \oplus M'_j$, as defined at line 43. Each of these Y'_{i-1} values with the exception of Y'_k was uniformly selected at line 41, independent of the now-determined domain of ι. As for $X'_{k+1} = Y'_k \oplus M'_{k+1}$, the value Y'_k was just selected uniformly at random back at line 12 and is independent of the domain of ι apart from the domain point $Y_k \oplus M_{k+1}$, which is *guaranteed* to be distinct from $Y'_k \oplus M'_{k+1}$ by our criterion for choosing k. Thus each time line 45 executes there is at most a $(m-k)/2^n$ chance that *bad* will get set, and the line is executed $m'-k$ times.

The probability that *bad* gets set at **line 46** of game C2 is at most $k(m'-k)$ for reasons exactly analogous to that argument used at line 24.

We conclude by the sum bound that the probability that *bad* gets set somewhere in game C2 is at most $(k(k-1)/2+k(m-k)+(m'-k)+(m-k)(m'-k)+k(m'-k))/2^n = mm'+m'-k(k-3)/2$. Since k can be zero, this value is at most $(mm'+m')/2^n$. Recalling that m' is the block length of the longer of M and M', the proof is complete. ▌

5.8 Collision-resistance under hidden-key attack

In a hidden-key attack, the adversary does not get the key K in the post-key attack phase, but instead gets an oracle for $H_K(\cdot)$. There are again three possible notions of security, analogous to those in Fig. 5.3 except that, in the post-key attack phase, A is not given K but is instead given an oracle for $H_K(\cdot)$. The CR0 notion however coincides with the one for known-key attacks since by the time the post-key attack phase is reached, a cr0-adversary has already output both its points, so we get only two new notions. Formal experiments defining these two notions are given in Fig. 5.9.

```
$\mathbf{Exp}_H^{\text{cr2-hk}}(A)$
    $K \xleftarrow{\$} \mathcal{K}$ ; Run $A^{H_K(\cdot)}()$
    If there exist $x_1, x_2$ such that
    –  $x_1 \neq x_2$ and $x_1, x_2 \in D$
    –  Oracle queries $x_1, x_2$ were made by $A$
    –  The answers returned by the oracle were the same
    then return 1 else return 0

$\mathbf{Exp}_H^{\text{cr1-hk}}(A)$
    $(x_1, st) \xleftarrow{\$} A()$ ; $K \xleftarrow{\$} \mathcal{K}$ ; Run $A^{H_K(\cdot)}(st)$
    If there exists $x_2$ such that
    –  $x_1 \neq x_2$ and $x_1, x_2 \in D$
    –  Oracle queries $x_1, x_2$ were made by $A$
    –  The answers returned by the oracle were the same
    then return 1 else return 0
```

Figure 5.9: Experiments defining security notions for two kinds of collision-resistant hash functions under hidden-key attack.

Definition 5.10 Let H: $\mathcal{K} \times D \to R$ be a hash function and let A be an algorithm. We let

$$\begin{aligned} \mathbf{Adv}_H^{\text{cr2-hk}}(A) &= \Pr\left[\mathbf{Exp}_H^{\text{cr2-hk}}(A) = 1\right] \\ \mathbf{Adv}_H^{\text{cr1-hk}}(A) &= \Pr\left[\mathbf{Exp}_H^{\text{cr1-hk}}(A) = 1\right] . \blacksquare \end{aligned}$$

the hash function.

5.9 Problems

Problem 5.1 Hash functions have sometimes been constructed using a block cipher, and often this has not gone well. Let E: $\mathcal{K} \times \{0,1\}^n \to \{0,1\}^n$ be a block cipher and consider constructing H: $\mathcal{K} \times (\{0,1\}^n)^+ \to \{0,1\}^n$ by way of the CBC construction: let the hash of $M_1 \cdots M_m$ be Y_m where $Y_0 = 0^n$ and $Y_i = E_K(H_{i-1} \oplus M_i)$ for $i \geq 1$. Here we select K to be some public constant. Show that this hash function is not collision-resistant (no matter how good is the block cipher E). $\blacksquare$

Problem 5.2 Let $H : \mathcal{K} \times \{0,1\}^a \to \{0,1\}^n$ be an ϵ-AU hash-function family. Construct from H an ϵ-AU hash-function family $H' : \mathcal{K} \times \{0,1\}^{2a} \to \{0,1\}^{2n}$. $\blacksquare$

Problem 5.3 Let $H : \mathcal{K} \times \{0,1\}^a \to \{0,1\}^n$ be an ϵ-AU hash-function family. Construct from H an ϵ^2-AU hash-function family $H' : \mathcal{K}^2 \times \{0,1\}^a \to \{0,1\}^{2n}$. $\blacksquare$

Bibliography

[1] M. Bellare and P. Rogaway. Collision-Resistant Hashing: Towards Making UOWHFs Practical. *Advances in Cryptology – CRYPTO '97*, Lecture Notes in Computer Science Vol. 1294, B. Kaliski ed., Springer-Verlag, 1997.

[2] M. Bellare and T. Kohno. Hash function balance and its impact on birthday attacks. *Advances in Cryptology – EUROCRYPT '04*, Lecture Notes in Computer Science Vol. 3027, C. Cachin and J. Camenisch ed., Springer-Verlag, 2004.

[3] I. Damgård. A Design Principle for Hash Functions. *Advances in Cryptology – CRYPTO '89*, Lecture Notes in Computer Science Vol. 435, G. Brassard ed., Springer-Verlag, 1989.

[4] B. den Boer and A. Bosselaers, Collisions for the compression function of MD5. *Advances in Cryptology – EUROCRYPT '93*, Lecture Notes in Computer Science Vol. 765, T. Helleseth ed., Springer-Verlag, 1993.

[5] H. Dobbertin, Cryptanalysis of MD4. *Fast Software Encryption—Cambridge Workshop*, Lecture Notes in Computer Science, vol. 1039, D. Gollman, ed., Springer-Verlag, 1996.

[6] H. Dobbertin, Cryptanalysis of MD5. Rump Session of Eurocrypt 96, May 1996, `http://www.iacr.org/conferences/ec96/rump/index.html`.

[7] H. Dobbertin, A. Bosselaers, B. Preneel. RIPEMD-160, a strengthened version of RIPEMD. *Fast Software Encryption '96*, Lecture Notes in Computer Science Vol. 1039, D. Gollmann ed., Springer-Verlag, 1996.

[8] M. Naor and M. Yung, Universal one-way hash functions and their cryptographic applications. *Proceedings of the 21st Annual Symposium on the Theory of Computing*, ACM, 1989.

[9] National Institute of Standards. FIPS 180-2, Secure hash standard, August 2000. `http://csrc.nist.gov/publications/fips/fips180-2/fips180-2.pdf`.

[10] R. Merkle. One way hash functions and DES. *Advances in Cryptology – CRYPTO '89*, Lecture Notes in Computer Science Vol. 435, G. Brassard ed., Springer-Verlag, 1989.

[11] R. Rivest, The MD4 message-digest algorithm, *Advances in Cryptology – CRYPTO '90*, Lecture Notes in Computer Science Vol. 537, A. J. Menezes and S. Vanstone ed., Springer-Verlag, 1990, pp. 303–311. Also IETF RFC 1320 (April 1992).

[12] R. Rivest, The MD5 message-digest algorithm. IETF RFC 1321 (April 1992).