2. n should not be too large since it takes $n2^n$ bits to specify a random function from $\mathcal{H}_n$. However, kn and hence k should be sufficiently large, otherwise the cipher is insecure even against the trivial exhaustive search attack.

3. When a relatively short key and a small n are chosen, the strength of the cipher will be significantly reduced. An effective method of resolving the problem is increasing the number of rounds of transformations.

The remaining part of this section proposes four example ciphers which we hope are secure enough for practical applications. Main parameters of the ciphers are collected in Table 4.1. For completeness, the definitions of the parameters are summarized below the table.

These parameters are chosen according to the preceding three aspects. In addition, $n = 4$ and $n = 8$ are chosen for easier implementation by software and/or hardware. The key-expanding part of each example cipher is realized by the algorithm G_m expanding a key of length $2m2^m$ bits into a long string of length $2m2^{2m}$ bits. All output bits of G_m are used by the enciphering part.

Notice that in Examples 2 and 3, the output of G_m is only half of the bits required by the enciphering part. We take two $2m2^m$-bit strings, and use G_m to stretch them into $2m2^{2m}$-bit ones. Then we combine the $2m2^{2m}$-bit strings into a $4m2^{2m}$-bit one. A recommended method for combining strings is concatenating them in bit/bits unit.

Definitions of Parameters

- Length of Plaintext/Ciphertext $= n \cdot k$ (bits).
- Length of Key $= t \cdot 2 \cdot m \cdot 2^m$ (bits), where t is the number of times G_m is applied.
- Size of Enciphering Part $= \ell \cdot s \cdot n \cdot 2^n$ (bits) $= \ell \cdot s \cdot n \cdot 2^n / 2^{13}$ (kilo-bytes).
- s — the number of rounds of Generalized Type-2 transformations applied in the enciphering/deciphering part.
- n — the length of a substring B_i (or C_i).
- k — $(= 2\ell)$ the number of substrings B_i's (or C_i's).
- m — specifying the length, $2m2^m$, of an input to G_m.

Table 4.1: Four Example Ciphers

Parameters	Example 1	Example 2	Example 3	Example 4
Length of Plaintext/ Ciphertext (bits)	64	96	112	128
Length of Key (bits)	768	1536 (=2·768)*	3584 (=2·1792)*	4096
Size of Enciphering Part (kilo-bytes)	6	12	56	128
s (rounds)	96	128	32	64
n (bits)	4	4	8	8
k	16	24	14	16
m	6	6	7	8

* (G_m is used twice)

Part II

Constructing One-Way Hash Functions

Chapter 5

Definitions

Part II of this thesis is concerned with one-way hash functions. One-way hash functions have many cryptographic applications. In digital signatures, they are used to compress long input strings prior to actual signing procedures. This usually greatly improves the overall efficiency of a signature scheme. They are also used to detect un-authorized modifications to important messages by such as malicious users or computer virus (See Figure 5.1 as suggested by Naor and Yung [NY89]). Another novel application of (provably secure) one-way hash functions, also due to Naor and Yung [NY89], is that they can be used to construct (provably secure) digital signature schemes.

There are roughly two kinds of one-way hash functions: *universal one-way hash functions* (UOHs) and *collision intractable hash functions* (CIHs). The main property of the former is that *given an initial-string* x, it is computationally difficult to find a different string y that collides with x. And the main property of the latter is that it is computationally difficult to find a pair $x \neq y$ of strings such that x collides with y. Formal definitions for these two kinds of one-way hash functions are given below. In Chapter 6 we reveal a duality between the construction of PSG and that of one-way hash functions. And in Chapter 7 we discuss structural properties of one-way hash functions.

5.1 One-Way Functions

Let ℓ be a monotone increasing function from $\mathcal{N}$ to $\mathcal{N}$, and f a (total) function from D to R, where $D = \bigcup_n D_n, D_n \subseteq \Sigma^n$, and $R = \bigcup_n R_n, R_n \subseteq \Sigma^{\ell(n)}$. D is called the *domain*, and R the *range* of f. In the following chapters, it is assumed, unless otherwise specified, that $D_n = \Sigma^n$ and $R_n = \Sigma^{\ell(n)}$. Denote by f_n the restriction of f on Σ^n. We are concerned only with the case when the range of f_n is $\Sigma^{\ell(n)}$, i.e., f_n is a function

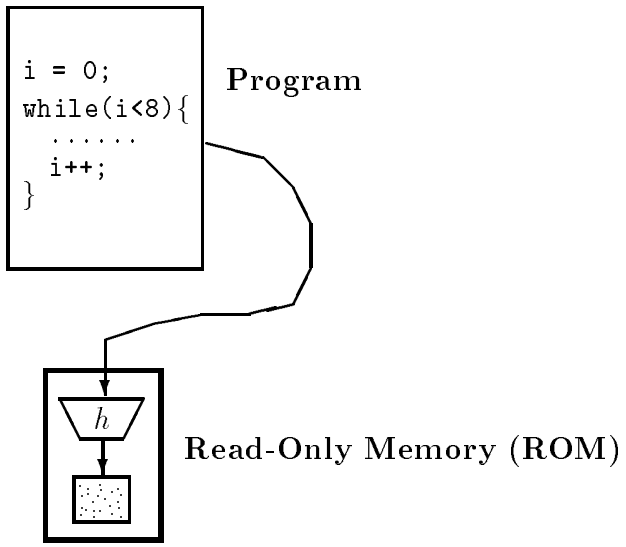


Figure 5.1: One-Way Hash Functions as Anti-Virus

from Σ^n to $\Sigma^{\ell(n)}$. f is an *injection* if each f_n is a one-to-one function, and is a *permutation* if each f_n is a one-to-one and onto function. f is (deterministic/probabilistic) *polynomial time computable* if there is a (deterministic/probabilistic) polynomial time algorithm (Turing machine) computing $f(x)$ for all $x \in D$.

A (probability) *ensemble* E with length $\ell(n)$ is a family of *probability distributions* $\{E_n | E_n : \Sigma^{\ell(n)} \rightarrow [0,1], n \in \mathcal{N}\}$. The *uniform ensemble* U with length $\ell(n)$ is the family of *uniform probability distributions* U_n, where each U_n is defined as $U_n(x) = 1/2^{\ell(n)}$ for all $x \in \Sigma^{\ell(n)}$. By $x \in_E \Sigma^{\ell(n)}$ we mean that x is randomly chosen from $\Sigma^{\ell(n)}$ according to E_n, and in particular, by $x \in_R S$ we mean that x is chosen from the set S uniformly at random. E is *samplable* if there is a (probabilistic) algorithm M that on input n outputs an $x \in_E \Sigma^{\ell(n)}$, and *polynomially samplable* if furthermore, the running time of M is polynomially bounded.

Let ℓ be a polynomial. A *uniform statistical test (for strings)* is a probabilistic polynomial time algorithm T that, on input a string x, outputs a bit 0/1. Let E^1 and E^2 be ensembles both with length $\ell(n)$. E^1 and E^2 are called *indistinguishable* from each other if for each uniform statistical test T, for each polynomial Q, for all sufficiently large n,

$$|\Pr\{T(x_1) = 1\} - \Pr\{T(x_2) = 1\}| < 1/Q(n),$$

where $x_1 \in_{E^1} \Sigma^{\ell(n)}, x_2 \in_{E^2} \Sigma^{\ell(n)}$. A polynomially samplable ensemble E is *pseudo-random* if it is indistinguishable from the uniform ensemble U with the same length.

One-way function is the basis of most of modern cryptographic functions and protocols [IL89]. The following definition is from [ILL89].

Definition 4 *Let $f : D \to R$, where $D = \bigcup_n \Sigma^n$ and $R = \bigcup_n \Sigma^{\ell(n)}$, be a polynomial time computable function, and let E be an ensemble with length n. We say that*

1. *f is* one-way with respect to *E if for each probabilistic polynomial time algorithm M, for each polynomial Q and for all sufficiently large n,*

$$\Pr\{f_n(x) = f_n(M(f_n(x)))\} < 1/Q(n),$$

   when $x \in_E D_n$.

2. *f is* one-way *if it is one-way with respect to the uniform ensemble U with length n.*

We note that a function f is one-way (with respect to the uniform ensemble U with length n) iff f is one-way with respect to *all* pseudo-random ensembles with the same length. This fact will be used in the proof of Theorem 20.

There are two basic computation models: Turing machines and combinational circuits (see for examples [Pip79, KL82, BDG88]). The above definition for one-way functions is with respect to the Turing machine model. A stronger version of one-way functions that is with respect to the circuit model (i.e., the non-uniform model) can be obtained by changing algorithms M in the above definition to families $M = \{M_n \mid n \in \mathcal{N}\}$ of polynomial size circuits.

Next we introduce the concept of (simultaneously) hard bits.

Definition 5 *Assume that $f : D \to R$ is a one-way function, where $D = \bigcup_n \Sigma^n$ and $R = \bigcup_n \Sigma^{\ell(n)}$. Also assume that $i_1, i_2, \ldots, i_t$ are functions from N to N, with $1 \le i_j(n) \le n$ for each $1 \le j \le t$. Denote by E_n^1 and E_n^2 the probability distributions defined by the random variables $x_{i_t(n)} \cdots x_{i_2(n)} x_{i_1(n)} \diamond f(x)$ and $r_t \cdots r_2 r_1 \diamond f(x)$ respectively, where $x \in_R \Sigma^n$, $x_{i_j(n)}$ is the $i_j(n)$-th bit of x and $r_j \in_R \Sigma$. Let $E^1 = \{E_n^1 \mid n \in \mathcal{N}\}$ and $E^2 = \{E_n^2 \mid n \in \mathcal{N}\}$. We say that*

1. *$i_1(n)$ is a* hard bit *of f if for each probabilistic polynomial time algorithm M, for each polynomial Q and for all sufficiently large n,*

$$\Pr\{M(n, f_n(x)) = x'_{i_1(n)}\} < 1/2 + 1/Q(n),$$

   where $x \in_R \Sigma^n$ and $x'_{i_1(n)}$ is the $i_1(n)$-th bit of an $x' \in \Sigma^n$ satisfying $f(x) = f(x')$.

2. *$i_1(n), i_2(n), \ldots, i_t(n)$ are* simultaneously *hard bits of f if E^1 and E^2 are indistinguishable from each other.*

5.2 Universal Hash Functions

Let ℓ be a polynomial with $\ell(n) > n$, H be a family of functions defined by $H = \bigcup_n H_n$ where H_n is a (possibly multi-)set of functions from $\Sigma^{\ell(n)}$ to Σ^n. Call H a *hash function (family)* compressing $\ell(n)$-bit input into n-bit output strings. For two strings $x, y \in \Sigma^{\ell(n)}$ with $x \neq y$, we say that x and y collide with each other under $h \in H_n$, or (x, y) is a collision pair for h, if $h(x) = h(y)$.

H is *polynomial time computable* if there is a polynomial (in n) time algorithm computing all $h \in H$, and *accessible* if there is a probabilistic polynomial time algorithm that on input $n \in \mathcal{N}$ outputs uniformly at random a description of $h \in H_n$. It is assumed that all hash functions considered here are both polynomial time computable and accessible.

Universal hash functions, first introduced in [CW79], play essential rules in many recent key results in cryptography [Has90, ILL89, Rom90] and theoretical computer science.

Definition 6 *Let k be a fixed positive integer, and H a hash function compressing $\ell(n)$-bit input into $m(n)$-bit output strings. H is a* (strong) universal$_k$ hash function *if for all n, for all k (distinct) strings $x_1, x_2, \ldots, x_k \in \Sigma^{\ell(n)}$ and all k strings $y_1, y_2, \ldots, y_k \in \Sigma^{m(n)}$, there are $\sharp H_n / 2^{km(n)}$ functions in H_n that map x_1 to y_1, x_2 to y_2, ..., x_k to y_k.*

Definition 7 *Let H be a (strong) universal$_k$ hash function compressing $\ell(n)$-bit input into $m(n)$-bit output strings. H has* the correspondence accessibility property [1] *if for all n, for all $1 \leq j \leq k$, for all (distinct) $x_1, x_2, \ldots, x_j \in \Sigma^{\ell(n)}$, and for all $y_1, y_2, \ldots, y_j \in \Sigma^{m(n)}$, it is possible in probabilistic polynomial time to uniformly sample from H'_n, where H'_n is the collection of all functions in H_n that map x_1 to y_1, x_2 to y_2, ..., x_j to y_j.*

Strong universal$_k$ hash functions with the correspondence accessibility property can be obtained from polynomials over finite fields [CW79, WC81]. Denote by P_n the collection of all polynomials over $GF(2^{\ell(n)})$ with degree less than k, i.e.,

$$P_n = \{a_0 + a_1 x + \cdots + a_{k-1} x^{k-1} | a_0, a_1, \ldots, a_{k-1} \in GF(2^{\ell(n)})\}.$$

For each $p \in P_n$, let h_p be the function that chooses the first $m(n)$ bits of the output of p. Let $H_n = \{h_p | p \in P_n\}$, and $H = \bigcup_n H_n$. Then H is a strong universal$_k$ hash function that compresses $\ell(n)$-bit input into $m(n)$-bit output strings and has the correspondence accessibility property.

[1] The *correspondence accessibility* property, introduced in [Rom90], is a generalization of the *collision accessibility* property defined in [NY89].

5.3 One-Way Hash Functions

As mentioned at the beginning of this chapter, there are basically two kinds of one-way hash functions: *universal one-way hash functions* and *collision-intractable hash functions* (or shortly UOHs and CIHs, respectively). In [Mer89] the former is called *weakly* and the latter *strongly*, one-way hash functions respectively. Naor and Yung gave a formal definition for UOH [NY89], and Damgård gave for CIH [Dam89].

5.3.1 Universal One-Way Hash Functions

Let H be a hash function compressing $\ell(n)$-bit input into n-bit output strings, and E an ensemble with length $\ell(n)$. The definition for UOH is best described as a three-party game. See also Figure 5.2. The three parties are S (an *initial-string supplier*), G (a *hash function instance generator*) and F (a *collision-string finder*). S is an oracle whose power is un-limited, and both G and F are probabilistic polynomial time algorithms. The first move is taken by S, who outputs an *initial-string* $x \in_E \Sigma^{\ell(n)}$ and sends it to both G and F. Also, S is allowed to provide F with all information on how the initial-string x is produced, if it is possible and requested by F. The second move is taken by G, who chooses, independently of x, an $h \in_R H_n$ and sends it to F. The third and also final (null) move is taken by F, who on input $x \in \Sigma^{\ell(n)}$ and $h \in H_n$, (as well as information on how x is produced,) outputs either "?" (I don't know) or a string $y \in \Sigma^{\ell(n)}$ such that $x \neq y$ and $h(x) = h(y)$. F wins a game iff his/her output is *not* equal to "?". Informally, H is a universal one-way hash function with respect to E if for any collision-string finder F, the probability that F wins a game is negligible. [2] , [3] More precisely:

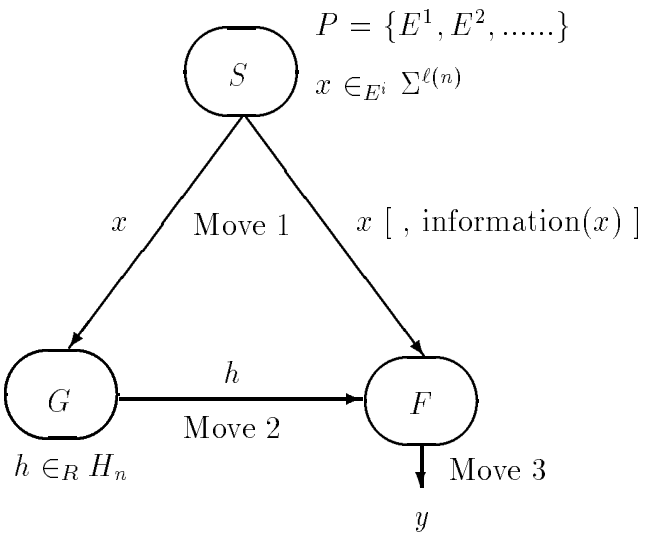
Definition 8 *Let H be a hash function compressing $\ell(n)$-bit input into n-bit output strings, P a collection of ensembles with length $\ell(n)$, and F a collision-string finder. H is a* universal one-way hash function with respect to P, *denoted by UOH/P, if for each $E \in P$, for each F, for each polynomial Q, and for all sufficiently large n,*

$$\Pr\{F(x,h) \neq ?\} < 1/Q(n),$$

[2] In [ZMI90b, ZMI90c], we did not explicitly state that *S is allowed to provide F with all information on how the initial-string x is produced.* We are grateful to John Leo, Kazuo Ohta and Kouichi Sakurai for pointing out potential problems, as shown in [Leo90], with this aspect.

[3] All collision-string finders discussed in this thesis are those which, when necessary and also possible, can receive from initial-string suppliers information on how initial-strings are produced. In most cases, this will not be explicitly indicated for the sake of simplicity. Such a finder is in general more powerful than a finder without receiving the information. This implies that UOHs defined in this thesis are in general stronger than those with respect to collision-string finders without receiving the information.

where x and h are independently chosen from $\Sigma^{\ell(n)}$ and H_n according to E_n and to the uniform distribution over H_n respectively, and the probability $\Pr\{F(x,h) \neq ?\}$ *is computed over $\Sigma^{\ell(n)}$, H_n and the sample space of all finite strings of coin flips that F could have tossed.*



information(x) — information on how x is produced.

Figure 5.2: UOH as a Three-Party Game

If P consists of a single ensemble E (i.e., $P = \{E\}$), UOH/E is synonymous with UOH/P. Of particular interest are the following versions of UOH:

1. UOH/$EN[\ell]$, where $EN[\ell]$ is the collection of all ensembles with length $\ell(n)$.
2. UOH/$PSE[\ell]$, where $PSE[\ell]$ is the collection of all polynomially samplable ensembles with length $\ell(n)$.
3. UOH/U, where U is the uniform ensemble with length $\ell(n)$.

In [NY89], Naor and Yung gave a definition for UOH. They did not separate initial-string ensembles from collision-string finders. Instead, they introduced a probabilistic *polynomial* time algorithm $A(\cdot,\cdot)$, called a *collision adversary* that works in two stages: At the first stage, the algorithm A, on input (λ,λ) where λ denotes the empty string, outputs an *initial value* (corresponding to our *initial-string*)

$x = A(\lambda, \lambda) \in \Sigma^{\ell(n)}$. At the second stage, it, when given an $h \in H_n$, attempts to find a string $y = A(x, h) \in \Sigma^{\ell(n)}$ such that $x \neq y$ and $h(x) = h(y)$.

Thus Naor and Yung defined, in our terms, *universal one-way hash function with respect to polynomially samplable ensembles with length* $\ell(n)$, i.e., UOH/$PSE[\ell]$. Naor and Yung constructed one-way hash functions in the sense of UOH/$PSE[\ell]$ under the assumption of the existence of one-way injections [NY89]. Note that they actually obtained a construction for one-way hash functions in the sense of UOH/$EN[\ell]$. In Chapter 6 we construct, in a different approach, one-way hash functions in the sense of UOH/$EN[\ell]$ under the assumption of the existence of one-way permutations.

Separating initial-string ensembles from collision-string finders is conceptually much clearer, and enables us to reduce the problem of constructing one-way hash functions in the sense of UOH/$EN[\ell]$ (the "strongest" UOHs) to that of constructing one-way hash functions in the sense of UOH/U (the "weakest" UOHs). This topic is treated in Chapter 7.

The above definition for UOH is with respect to the Turing machine model. As a natural counterpart of UOH/P, where P is a set of ensembles with length $\ell(n)$, we have UOH_C/P, whose definition is obtained simply by changing probabilistic polynomial time algorithms F in Definition 8 to families $F = \{F_n \mid n \in \mathcal{N}\}$ of polynomial size circuits.

The definition for UOH can also be generalized in another direction: In addition to $x \in \Sigma^{\ell(n)}$ and $h \in H_n$, a collision-string finder F is allowed to receive an extra *advice* string a. As before, the output of F is either "?" or a string $y \in \Sigma^{\ell(n)}$ such that $x \neq y$ and $h(x) = h(y)$.

Definition 9 *Let H be a hash function compressing $\ell(n)$-bit input into n-bit output strings. H is a* universal one-way hash function with respect to polynomial length advice, *denoted by UOH/EN*[poly], *if for each pair (Q_1, Q_2) of polynomials with $Q_1(n) \geq \ell(n)$, for each ensemble E with length $Q_1(n)$, for each collision-string finder F, and for all sufficiently large n,*

$$\Pr\{F(x, a, h) \neq ?\} < 1/Q_2(n),$$

where $x \in \Sigma^{\ell(n)}$, $a \in \Sigma^{Q_1(n)-\ell(n)}$, $x \diamond a$ and h are independently chosen from $\Sigma^{Q_1(n)}$ and H_n according to E_n and to the uniform distribution over H_n respectively, and the probability $\Pr\{F(x, a, h) \neq ?\}$ is computed over $\Sigma^{Q_1(n)}$, H_n and the sample space of all finite strings of coin flips that F could have tossed.

Five versions of UOH introduced above are shown in Table 5.1.

Notice the difference between *Turing machines taking advice* discussed in [Pip79, KL82] and collision-string finders in our Definition 9. In the former case, advice

Table 5.1: Classification of UOH

Versions of UOH	Initial-Strings Ensemble(s)	Models of Finders
UOH/U	U	Turing Machine
UOH/$PSE[\ell]$	$PSE[\ell]$	Turing Machine
UOH/$EN[\ell]$	$EN[\ell]$	Turing Machine
UOH$_C$/$EN[\ell]$	$EN[\ell]$	Circuit
UOH/EN[poly]	EN[poly]	Turing Machine

strings are uniquely determined for each $n \in \mathcal{N}$. While in the latter case, they are generated probabilistically. In Chapter 7, we will discuss relationships among various versions of one-way hash functions including UOH/U, UOH/$PSE[\ell]$, UOH/$EN[\ell]$, UOH$_C$/$EN[\ell]$, and UOH/EN[poly].

5.3.2 Collision Intractable Hash Functions

This section gives formal definitions for collision intractable hash functions. Let $H = \bigcup_n H_n$ be a hash function compressing $\ell(n)$-bit input into n-bit output strings. Let A, a *collision-pair finder*, be a probabilistic polynomial time algorithm that on input $h \in H_n$ outputs either "?" or a pair of strings $x, y \in \Sigma^{\ell(n)}$ with $x \neq y$ and $h(x) = h(y)$.

Definition 10 *H is called a* collision-intractable hash function (CIH) *if for each A, for each polynomial Q, and for all sufficiently large n,*

$$\Pr\{A(h) \neq ?\} < 1/Q(n),$$

where $h \in_R H_n$, and the probability $\Pr\{A(h) \neq ?\}$ *is computed over H_n and the sample space of all finite strings of coin flips that A could have tossed.*

In [Dam89] (see also [Dam87]) CIH is called *collision free function family*. Damgård obtained CIHs under the assumption of the existence of claw-free pairs of permutations. In Chapter 6 we propose *practical* CIHs, the fastest of which compress nearly $2n$-bit long input into n-bit long output strings by applying only *twice* a one-way function.

CIH defined above are with respect to the Turing machine model. So as in the case for UOH, we have CIH$_C$ with respect to the circuit model. The definition for

CIH_C is similar to Definition 10, except that probabilistic polynomial time algorithms A are replaced by families $A = \{A_n \mid n \in \mathcal{N}\}$ of polynomial size circuits.

In addition, analogous to Definition 9, we have the following generalization for CIH. Let $H = \bigcup_n H_n$ be a hash function compressing $\ell(n)$-bit input into n-bit output strings, Q_1 a polynomial, and $a \in \Sigma^{Q_1(n)}$. a is called an *advice* string of length $Q_1(n)$. Let A, a collision-pair finder, be a probabilistic polynomial time algorithm that on input $a \in \Sigma^{Q_1(n)}$ and $h \in H_n$ outputs either "?" or a pair of strings $x, y \in \Sigma^{\ell(n)}$ with $x \neq y$ and $h(x) = h(y)$.

Definition 11 *H is called a* collision intractable hash function with respect to polynomial length advice, *denoted by CIH/EN*[poly], *if for each pair (Q_1, Q_2) of polynomials, for each ensemble E with length $Q_1(n)$, for each A, and for all sufficiently large n,*

$$\Pr\{A(a, h) \neq ?\} < 1/Q_2(n),$$

where a and h are independently chosen from $\Sigma^{Q_1(n)}$ and H_n according to E_n and to the uniform distribution over H_n respectively, and the probability $\Pr\{A(a, h) \neq ?\}$ is computed over $\Sigma^{Q_1(n)}$, H_n and the sample space of all finite strings of coin flips that A could have tossed.

The three versions of CIH are summarized in Table 5.2.

Table 5.2: Classification of CIH

Versions of CIH	Models of Finders
CIH	Turing Machine
CIH_C	Circuit
CIH/EN[poly]	Turing Machine

Chapter 6

Duality between Two Cryptographic Primitives

This chapter reveals a duality between constructions of two basic cryptographic primitives, *pseudo-random string generators* and *one-way hash functions*. Applying the duality, we present a construction for *universal one-way hash functions* assuming the existence of one-way permutations. Using ideas behind the construction, we propose *practical* one-way hash functions, the fastest of which compress nearly $2n$-bit long input into n-bit long output strings by applying only *twice* a one-way function.

6.1 Extension and Compression Methods

In this section we discuss a duality between the construction of pseudo-random string generators (PSGs) and that of one-way hash functions. Throughout this Chapter, PSGs are polynomially secure ones (See Definition 1), t and s are integers, $\ell_0, \ell_1, \ldots, \ell_s$ are polynomials in n with $\ell_0(n) = n$ and $\ell_i(n) < \ell_{i+1}(n)$, and k is a polynomial in n such that $t \cdot k(n) > n$. The product of the polynomials t and k is denoted by ℓ.

6.1.1 Serial Versions

Two extension and two compression methods which are serial in nature are introduced below. Lemma 5 (serial-extension 1) is the dual of Lemma 7 (serial-compression 1), and Lemma 6 (serial-extension 2) is that of Lemma 8 (serial-compression 2).

Serial Extension Lemmas

For each $0 \leq i \leq s-1$, let $g^i = \{g^i_n \mid n \in \mathcal{N}\}$ be a PSG extending $\ell_i(n)$-bit input into $\ell_{i+1}(n)$-bit output strings. The following lemma is a direct consequence of the

definition for PSG. See also Figure 6.1.

Lemma 5 (serial-extension 1) *Let* $g = \{g_n = g_n^{s-1} \circ g_n^{s-2} \circ \cdots \circ g_n^0 \mid n \in \mathcal{N}\}$. *Then* g *is a PSG extending* n*-bit input into* $\ell_s(n)$*-bit output strings.*

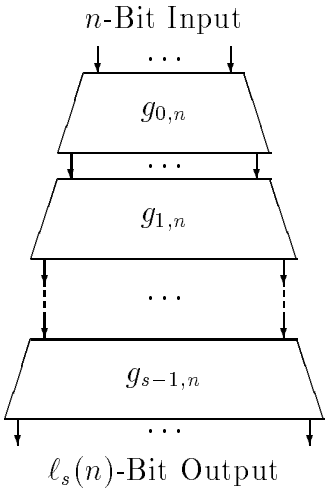


Figure 6.1: Serial-Extension 1

A PSG extending n-bit input into $(n+t)$-bit output strings is called a t-extender. Let y be a finite length string. Denote by $\text{head}_i(y)$ the first i bits and by $\text{tail}_i(y)$ the last i bits of y. The following lemma is due to Boppana and Hirschfeld [BH89]. See also Figure 6.2.

Lemma 6 (serial-extension 2) *Let* $e = \{e_n \mid n \in \mathcal{N}\}$ *be a* t*-extender. For an* n*-bit string* x*, let*

$$b_i(x) = \text{head}_t \circ e_n \circ (\text{tail}_n \circ e_n)^{(i-1)}(x),$$

where $1 \leq i \leq k(n)$. *Let* g_n *be the function defined by* $g_n(x) = b_{k(n)}(x) \diamond \cdots \diamond b_2(x) \diamond b_1(x)$. *Then* $g = \{g_n \mid n \in \mathcal{N}\}$ *is a PSG extending* n*-bit input into* $\ell(n)$*-bit output strings.*

Serial Compression Lemmas

For each $1 \leq i \leq s$, let $H^i = \bigcup_n H_n^i$ be a UOH/$EN[\ell_i]$ (or CIH) compressing $\ell_i(n)$-bit input into $\ell_{i-1}(n)$-bit output strings. Naor and Yung proved the following serial compression lemma [NY89]. See Figure 6.3.

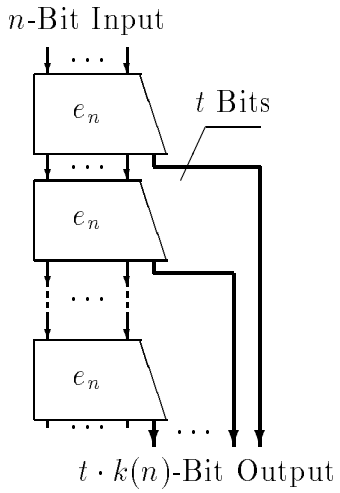


Figure 6.2: Serial-Extension 2

Lemma 7 (serial-compression 1) *Let $H = \bigcup_n H_n$, where*

$$H_n = \{h = h_1 \circ h_2 \circ \cdots \circ h_s \mid h_i \in H_n^i, 1 \leq i \leq s\}.$$

Then H is a UOH/$EN[\ell_s]$ (or CIH) compressing $\ell_s(n)$-bit input into n-bit output strings.

Let $\ell'(n) = n + t$, and let $C = \bigcup_n C_n$ be a UOH/$EN[\ell']$ (or CIH) compressing $\ell'(n)$-bit input into n-bit output strings. Such a UOH/$EN[\ell']$ (or CIH) is called a t-compressor. Then we have the following lemma that is a restricted version of Theorem 3.1 of [Dam89]. The main idea behind the lemma also appeared in [Mer89], where it was called the "meta-method". See Figure 6.4.

Lemma 8 (serial-compression 2) *Let $C = \bigcup_n C_n$ be a t-compressor. For each $c \in C_n$ and each $\alpha \in \Sigma^n$, let $h_{c,\alpha}$ be the function defined by*

$$h_{c,\alpha}(x) = c(\cdots(c(\alpha \diamond b_{k(n)}(x)) \diamond b_{k(n)-1}(x)) \cdots \diamond b_1(x))$$

where $x = x_{\ell(n)} \cdots x_2 x_1$ is an $\ell(n)$-bit string and $b_i(x) = x_{t(i-1)+t} \cdots x_{t(i-1)+2} x_{t(i-1)+1}$. Let $H_n = \{h_{c,\alpha} | c \in C_n, \alpha \in \Sigma^n\}$, and $H = \bigcup_n H_n$. Then H is a UOH/$EN[\ell]$ (or CIH) compressing $\ell(n)$-bit input into n-bit output strings.

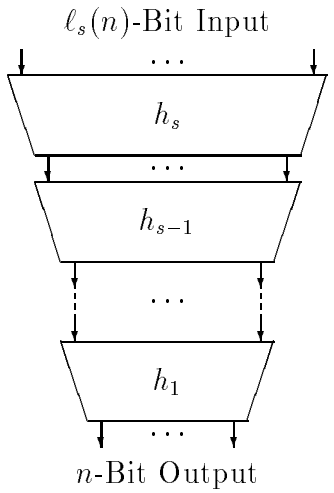


Figure 6.3: Serial-Compression 1

6.1.2 Parallel Versions

In their nice paper [GGM86], Goldreich et al. presented a method for constructing *pseudo-random functions* from pseudo-random string generators. Their construction provides us a configuration for generating pseudo-random strings in parallel, when given polynomially many processors. This observation is the very basis of Micali and Schnorr's parallel PSGs [MSch88]. On the other hand, a parallel hashing method was considered in several papers such as [WC81, NY89, Dam89]. Duality between these two methods is clear. Details are omitted here.

6.2 PSGs and UOHs from One-Way Permutations

Throughout this section, it is assumed that f is a one-way permutation on $D = \bigcup_n \Sigma^n$, and that $i(n)$ has been proved to be a hard bit of f.

6.2.1 PSGs from One-Way Permutations

Denote by $\mathrm{extract}_i(x)$ a function extracting the i-th bit of $x \in \Sigma^n$. The following theorem is due to Blum and Micali [BM84]. See also Figure 6.5.

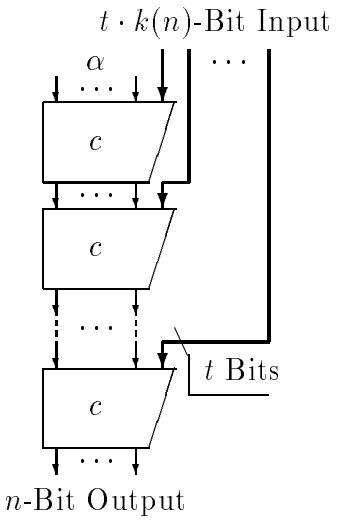


Figure 6.4: Serial-Compression 2

Theorem 19 *Let ℓ be a polynomial with $\ell(n) > n$, and let g_n be the function defined by $g_n(x) = b_{\ell(n)}(x) \cdots b_2(x) b_1(x)$ where $x \in \Sigma^n$ and $b_j(x) = \mathrm{extract}_{i(n)}(f_n^{(j)}(x))$. Then under the assumption that f is a one-way permutation, $g = \{g_n | n \in \mathcal{N}\}$ is a PSG extending n-bit input into $\ell(n)$-bit output strings.*

The efficiency of g can be improved by changing $\mathrm{extract}_{i(n)}()$ to a function that extracts all known simultaneously hard bits of f.

6.2.2 UOHs from One-Way Permutations

For $b \in \Sigma$, $x \in \Sigma^{n-1}$ and $y \in \Sigma^n$, define $\mathrm{ins}_i(x, b) = x_{n-1} x_{i-2} \cdots x_i b x_{i-1} \cdots x_2 x_1$, and denote by $\mathrm{drop}_i(y)$ a function dropping the i-th bit of y. As the dual of Theorem 19, we have the following result. See also Figure 6.6.

Theorem 20 *Let ℓ be a polynomial with $\ell(n) > n$, $\alpha \in \Sigma^{n-1}$ and $x = x_{\ell(n)} \cdots x_2 x_1$ where $x_i \in \Sigma$ for each $1 \leq i \leq \ell(n)$. Let h_α be the function from $\Sigma^{\ell(n)}$ to Σ^n defined by:*

$$
\begin{aligned}
y_0 &= \alpha, \\
y_1 &= \mathrm{drop}_{i(n)}(f_n(\mathrm{ins}_{i(n)}(y_0, x_{\ell(n)}))), \\
&\cdots \\
y_j &= \mathrm{drop}_{i(n)}(f_n(\mathrm{ins}_{i(n)}(y_{j-1}, x_{\ell(n)-j+1}))),
\end{aligned}
$$

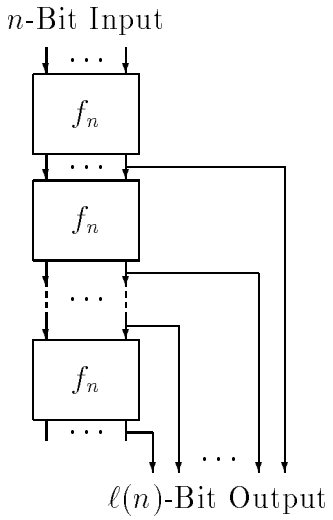


Figure 6.5: PSG from a One-Way Permutation

$$\begin{aligned} & \cdots \\ h_\alpha(x) \quad &= \quad f_n(\mathrm{ins}_{i(n)}(y_{\ell(n)-1}, x_1)). \end{aligned}$$

Let $H_n = \{h_\alpha \mid \alpha \in \Sigma^{n-1}\}$ and $H = \bigcup_n H_n$. Then under the assumption that f is a one-way permutation, H is a UOH/EN[ℓ] compressing $\ell(n)$-bit input into n-bit output strings.

Proof : Assume that E is an initial-string ensemble and F a collision-string finder. Let $x \in_E \Sigma^{\ell(n)}$. We show that if F finds with probability $p(n)$ a string x' such that $h(x) = h(x')$ where $h \in_R H_n$, then there is a probabilistic polynomial time algorithm M that finds, with probability greater than $p(n)/(\ell(n) - 1)$, the inverse $f_n^{-1}(w)$ of an n-bit string $w \in_T \Sigma^n$, where T is a pseudo-random ensemble.

The proving procedure consists of three parts. In the first part, we show that every execution of f_n in h defines two pseudo-random ensembles S^j and R^j, if $\alpha \in_R \Sigma^{n-1}$. From each S^j we construct another pseudo-random ensemble T^j. In the second part, we construct a probabilistic polynomial time algorithm M. M first obtains $w \in_T \Sigma^n$, where T is an ensemble chosen uniformly at random from all ensembles T^j. Then it computes the inverse $f_n^{-1}(w)$ by the use of the collision-string finder F. In the third part we estimate the probability that M outputs $f_n^{-1}(w)$ correctly.

Let S_n^1 be the probability distribution defined by $f_n(\mathrm{ins}_{i(n)}(\alpha, x_{\ell(n)}))$, where $\alpha \in_R \Sigma^{n-1}$ and $x_{\ell(n)}$ is the last bit of $x \in_E \Sigma^{\ell(n)}$. Let $S^1 = \{S_n^1 \mid n \in \mathcal{N}\}$. Clearly S^1 is polynomi-

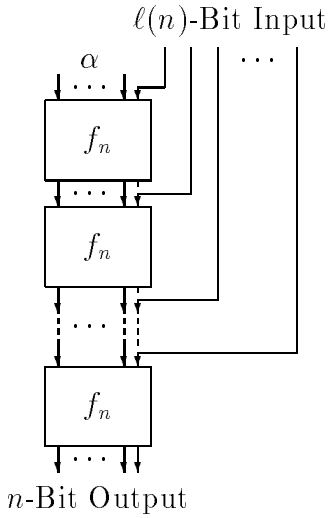


Figure 6.6: UOH/$EN[\ell]$ from a One-Way Permutation

ally samplable (when $x \in_E \Sigma^{\ell(n)}$ is given). Now we show that S^1 is indistinguishable from the uniform ensemble.

Let A be a probabilistic polynomial time algorithm. For $v \in \Sigma$, denote by $\Pr^v$ the probability that A outputs 1 on input $f_n(\mathrm{ins}_{i(n)}(\alpha, v))$. Since $\alpha \in_R \Sigma^{n-1}$ and $i(n)$ is a hard bit of f (by assumption), we can think of $z = f_n(\mathrm{ins}_{i(n)}(\alpha, x_{\ell(n)}))$ as a probabilistic encryption of $x_{\ell(n)}$ [GM84]. Thus for any probabilistic polynomial time algorithm A, for any polynomial Q, for all sufficiently large n, we have $|\Pr^0 - \Pr^1| < 1/Q(n)$. Now let $v \in_R \Sigma$. Then

$$\begin{aligned} \Pr^v &= \Pr^0 \cdot \Pr\{v = 0\} + \Pr^1 \cdot \Pr\{v = 1\} \\ &= (\Pr^0 + \Pr^1)/2, \end{aligned}$$

and hence

$$\begin{aligned} |\Pr^v - \Pr^0| &= |\Pr^v - \Pr^1| \\ &= |\Pr^0 - \Pr^1|/2 \\ &< 1/2Q(n). \end{aligned}$$

This implies that

$$|\Pr^v - \Pr^{x_{\ell(n)}}| < 1/2Q(n),$$

no matter how $x_{\ell(n)}$ is chosen from Σ. Note that when $\alpha \in_R \Sigma^{n-1}$ and $v \in_R \Sigma$, we have $f_n(\mathrm{ins}_{i(n)}(\alpha, v)) \in_R \Sigma^n$, since f is a permutation. From these discussions, we see that S^1 is indeed pseudo-random.

Let $R^1 = \{R^1_n \mid n \in \mathcal{N}\}$, where R^1_n is defined by $\mathrm{drop}_{i(n)}(f_n(\mathrm{ins}_{i(n)}(\alpha, x_{\ell(n)})))$. Then R^1 is also pseudo-random.

Let $S^2 = \{S^2_n \mid n \in \mathcal{N}\}$, where S^2_n is defined by $f_n(\mathrm{ins}_{i(n)}(\beta, x_{\ell(n)-1}))$, $\beta \in_{R^1} \Sigma^{n-1}$ and $x_{\ell(n)-1}$ is the second last bit of $x \in_E \Sigma^{\ell(n)}$. Then S^2 is also pseudo-random, for otherwise there would be a statistical test distinguishing R^1 from the uniform ensemble with length $n-1$. Similarly, for each $2 < j \leq \ell(n)$, we can define S^j, R^j and prove that they are both pseudo-random.

For each S^j, $1 \leq j \leq \ell(n) - 1$, define T^j as follows: $w \in_{T^j} \Sigma^n$ is produced by first generating a string $w' \in_{S^j} \Sigma^n$, then reversing the $i(n)$-th bit of w', i.e., $w = w' \oplus 0^{n-i(n)}10^{i(n)-1} = f_n(\mathrm{ins}_{i(n)}(y_{j-1}, x_{\ell(n)-j+1})) \oplus 0^{n-i(n)}10^{i(n)-1}$, where 0^i denotes the all-0 string of length i and $\oplus$ the bit-wise exclusive-or operation on two strings. Obviously, T^j is also pseudo-random.

We are ready to describe the algorithm M. Let O be an oracle that on input $n \in \mathcal{N}$ outputs a string $x \in_E \Sigma^{\ell(n)}$.

Algorithm M:

1. Choose $\alpha \in_R \Sigma^{n-1}$.
2. Query the oracle O with n. Let the answer by O be x. Note that $x \in_E \Sigma^{\ell(n)}$.
3. Choose at random a $1 \leq k \leq \ell(n) - 1$, and let $T = T^k$.
4. Let $w = f_n(\mathrm{ins}_{i(n)}(y_{k-1}, x_{\ell(n)-k+1})) \oplus 0^{n-i(n)}10^{i(n)-1}$, i.e., choose a $w \in_T \Sigma^n$.
5. Query the collision-string finder F with n, h and x. If F finds a string x' such that $h(x) = h(x')$, then output $\mathrm{ins}_{i(n)}(y'_{k-1}, x'_{\ell(n)-k+1})$, where y'_{k-1} and $x'_{\ell(n)-k+1}$ are defined similarly to y_{k-1} and $x_{\ell(n)-k+1}$ respectively. Otherwise, output a $z \in_R \Sigma^n$.

The running time of M is clearly bounded by a polynomial in n. Now we estimate the probability that the algorithm M outputs $f_n^{-1}(w)$ correctly. Note that when F finds an x' such that $h(x) = h(x')$, then there is an $1 \leq m \leq \ell(n) - 1$ such that $\mathrm{ins}_{i(n)}(y_{m-1}, x_{\ell(n)-m+1}) \neq \mathrm{ins}_{i(n)}(y'_{m-1}, x'_{\ell(n)-m+1})$ and $\mathrm{ins}_{i(n)}(y_{j-1}, x_{\ell(n)-j+1}) = \mathrm{ins}_{i(n)}(y'_{j-1}, x'_{\ell(n)-j+1})$ for each $m < j \leq \ell(n)$. Since k is chosen independently of F, the probability that $k = m$ is $1/(\ell(n)-1)$. So the probability that M outputs $f_n^{-1}(w)$ correctly is

$$\begin{aligned} &\Pr\{M \text{ outputs } f_n^{-1}(w)\} \\ &\quad > \Pr\{M \text{ outputs } f_n^{-1}(w) \mid F \text{ finds } x'\} \cdot \end{aligned}$$

$$\begin{aligned} & \Pr\{F \text{ finds } x'\} \\ = \; & p(n)/(\ell(n)-1). \end{aligned}$$

When $p(n) \geq 1/Q(n)$ for some polynomial Q, i.e., H is not a one-way hash function in the sense of UOH/$EN[\ell]$, we have $\Pr\{M \text{ outputs } f_n^{-1}(w)\} > 1/Q'$ where Q' is the polynomial defined by $Q'(n) = Q(n)(\ell(n)-1)$. In other words, f is not one-way with respect to the pseudo-random ensemble T, hence not one-way with respect to the uniform ensemble U (see the note following Definition 4). This is a contradiction and the proof is completed. □

Now let $I(n) = \{i_1, i_2, \ldots, i_{t(n)}\}$ be known simultaneously hard bits of f with $t(n) = O(\log n)$. Let $b = b_{t(n)} \cdots b_2 b_1 \in \Sigma^{t(n)}$, $x \in \Sigma^{n-t(n)}$ and $y \in \Sigma^n$. Define

$$\mathrm{ins}_{I(n)}(x,b) = x_{n-t(n)} \cdots x_{i_{t(n)}} b_{t(n)} x_{i_{t(n)}-1} \cdots x_{i_1} b_1 x_{i_1-1} \cdots x_2 x_1,$$

and denote by $\mathrm{drop}_{I(n)}(y)$ a function dropping the i_1-th, i_2-th, ..., $i_{t(n)}$-th bits of y. Then by changing $\mathrm{drop}_{i(n)}()$ to $\mathrm{drop}_{I(n)}()$, and x_i to

$$x_{t(n)(i-1)+t(n)} \cdots x_{t(n)(i-1)+2} x_{t(n)(i-1)+1},$$

the above constructed H is improved to a hash function that compresses $t(n)\ell(n)$-bit input into n-bit output strings. In other words, the efficiency of H can be improved by a factor of $O(\log n)$.

6.3 Practical One-Way Hash Functions

In this section we show that ideas underlying Theorems 20 can be used to design *practical* one-way hash functions. The fastest of these hash functions compress nearly $2n$-bit long input into n-bit long output strings by applying only *twice* a one-way function.

Let $f : D \to R$ be a one-way function where $D = \bigcup_n \Sigma^n$, $R = \bigcup_n \Sigma^{m(n)}$ and m is a polynomial. Let k be a real with $k > 1$, s an integer with $s \geq 2$ and $\ell(n) = s(n - \lfloor n/k \rfloor)$. Typically, we choose $1 < k \leq 10$ and $2 \leq s \leq 5$. For each $\alpha \in \Sigma^{\lfloor n/k \rfloor}$, associate with it a function h_α defined by:

$$\begin{aligned} y_0 &= \alpha, \\ y_1 &= \mathrm{tail}_{\lfloor n/k \rfloor}(f_n(y_0 \diamond x^s)), \\ &\cdots \\ y_j &= \mathrm{tail}_{\lfloor n/k \rfloor}(f_n(y_{j-1} \diamond x^{s-j+1})), \\ &\cdots \\ h_\alpha(x) &= f_n(y_{s-1} \diamond x^1). \end{aligned}$$

where $x = x^s \cdots x^2 x^1 \in \Sigma^{\ell(n)}$ and $x^1, x^2, \ldots, x^s \in \Sigma^{n-\lfloor n/k \rfloor}$. See also Figure 6.7. Let $H_n = \{h_\alpha | \alpha \in \Sigma^{\lfloor n/k \rfloor}\}$ and $H = \bigcup_n H_n$.

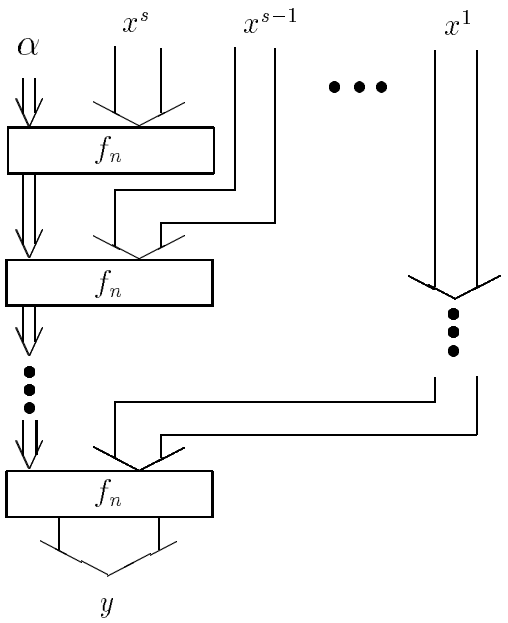


Figure 6.7: Practical Hashing Method

In practice, we first choose, uniformly at random, a string α from $\Sigma^{\lfloor n/k \rfloor}$, and fix it. Then by using the function h_α as is defined above, we can compress $\ell(n)$-bit input into n-bit output strings. This procedure is called the *Hashing Method*, and can be used as a basic step of the serial compression method defined in Lemma 8 and the parallel compression method mentioned in Section 6.1.2.

We were not able to prove that the hash function H is a CIH or a UOH/$EN[\ell]$, even under the assumption that 1 up to $n - \lfloor n/k \rfloor$ are all simultaneously hard bits of f. A sufficient condition for H to be a CIH is that *it is computationally difficult to find two distinct strings* $x, y \in \Sigma^n$ *such that* $\mathrm{tail}_{\lfloor n/k \rfloor}(f_n(x)) = \mathrm{tail}_{\lfloor n/k \rfloor}(f_n(y))$. A (secure) public-key encryption function can be viewed as a function satisfying the condition. For a common-key block cipher, the function from its key space to ciphertext space induced by a randomly chosen plaintext can also be viewed as a function satisfying the condition.

It seems that, when f is carefully chosen, H is *strong* enough and also *efficient* enough for practical applications. In the remaining portion of this section, we present two concrete examples. One is based on the Rabin encryption function, and the other

on a common-key block cipher called xDES. There is another good example based on the RSA encryption function with low exponents. Discussions for it are analogous to the first example, and hence omitted here.

6.3.1 Compressing via the Rabin Encryption Function

Let $M_n = pq$ where p and q are $n/2$-bit long randomly generated primes. Denote by Z_{M_n} the residue classes of integers modulo M_n. The Rabin encryption function *rabin* is defined by

$$rabin_n(x) = x^2 \bmod M_n$$

where $x \in Z_{M_n}$. For Blum integers M_n, i.e., $p \equiv q \equiv 3 \pmod 4$, it was proved that 1 up to $O(\log n)$ are simultaneously hard bits of *rabin*. Now let $k = 10$. When n is large (say ≥ 500), as the authors know, no currently existing algorithms can efficiently find two distinct elements $x, y \in Z_{M_n}$ such that the last $n/10$ bits of $x^2 \bmod M_n$ coincide with that of $y^2 \bmod M_n$.

By the use of the Rabin encryption function and the Hashing Method, we can compress 900-bit input to 500-bit output strings by performing only *twice* the multiplication of two 500-bit integers modulo an integer of the same length. This procedure can be implemented very efficiently, even by software.

6.3.2 Compressing via xDES

Let m be a polynomial. Informally, a common-key block cipher with length $m(n)$ is a pair of polynomial time computable functions $(encrypt, decrypt)$, where *encrypt* and *decrypt* are functions from $\bigcup_n \Sigma^n \times \Sigma^{m(n)}$ to $\bigcup_n \Sigma^{m(n)}$ that have the following properties:

1. $ptxt = decrypt_n(key, encrypt_n(key, ptxt))$ for all $key \in \Sigma^n$ and all $ptxt \in \Sigma^{m(n)}$.

2. It is computationally difficult to find $ptxt$ from $encrypt_n(key, ptxt)$ for any $ptxt \in \Sigma^{m(n)}$, without knowing key.

Let f be the function from $\bigcup_n \Sigma^n$ to $\bigcup_n \Sigma^{m(n)}$ that is defined by $f_n(x) = encrypt_n(x, \alpha)$, where $\alpha \in_R \Sigma^{m(n)}$. Then each f_n can be used to compress strings by the Hashing Method. To prevent the compression method from *meet-in-the-middle attacks*, n should be chosen in such a way that $m(n)$ is sufficiently large, say > 120. A rigorous treatment of this subject can be found in [NS90].

Consider the perhaps most widely used modern encryption algorithm DES. According to our definition, DES is the restriction of *some* common-key block cipher

on $\Sigma^{56} \times \Sigma^{64}$. DES should not be directly used to compress strings by the Hashing Method, for its key length is too short. Now we use DES as bricks to build a common-key block cipher called xDES. Our building method is based on the theory on the construction of secure block ciphers developed in Part I.

Let r be a polynomial with $r(i) \geq 2i + 1$. xDES is defined by xDES^0, xDES^1, xDES^2, xDES^3, $\cdots$, where xDES^0 is the same as DES and, for each $i \geq 1$, xDES^i is a function from $\Sigma^{56r(i)i} \times \Sigma^{128i}$ to Σ^{128i} consisting of $r(i)$ rounds of *Type-2 transformations* (See Section 3.1.2). More details follow.

1. The definition for xDES^0: Same as DES.

2. The definition for xDES^i where $i \geq 1$: Let

$$\begin{aligned} key \;=\; & key_{r(i),i} \diamond \cdots \diamond key_{r(i),2} \diamond key_{r(i),1} \diamond \\ & \cdots \diamond \\ & key_{2,i} \diamond \cdots \diamond key_{2,2} \diamond key_{2,1} \diamond \\ & key_{1,i} \diamond \cdots \diamond key_{1,2} \diamond key_{1,1} \end{aligned}$$

and

$$ptxt = ptxt_{2i} \diamond \cdots \diamond ptxt_2 \diamond ptxt_1,$$

where $key_{i_1,i_2} \in \Sigma^{56}$ and $ptxt_{i_3} \in \Sigma^{64}$ for all $1 \leq i_1 \leq r(i)$, $1 \leq i_2 \leq i$ and $1 \leq i_3 \leq 2i$. Then $ctxt = \text{xDES}^i(key, ptxt)$ is computed as follows:

Step 0:

$$\begin{aligned} c_{0,1} &= ptxt_1, \\ c_{0,2} &= ptxt_2, \\ &\cdots \\ c_{0,2i-1} &= ptxt_{2i-1}, \\ c_{0,2i} &= ptxt_{2i}, \end{aligned}$$

Step j, for each $1 \leq j \leq r(i)$:

$$\begin{aligned} c_{j,1} &= c_{j-1,2i}, \\ c_{j,2} &= c_{j-1,1} \oplus \text{DES}(key_{j,1}, c_{j-1,2}), \\ &\cdots \end{aligned}$$

$$
\begin{aligned}
c_{j,2i-1} &= c_{j-1,2i-2}, \\
c_{j,2i} &= c_{j-1,2i-1} \oplus \mathrm{DES}(key_{j,i}, c_{j-1,2i}),
\end{aligned}
$$

Step $r(i)+1$:

$$
ctxt = c_{r(i),2i} \diamond \cdots \diamond c_{r(i),2} \diamond c_{r(i),1}.
$$

Let $r(i) = 2i+1$ and $k = 3$. Using the Hashing Method, we can compress 224-bit input into 128-bit output strings by performing only twice xDES[1], i.e., 6 times DES. It should be noted that xDES can also be used in normal encryption/decryption operations, and DES can be replaced by any other secure common-key block encryption algorithm.

Table 6.1 summarizes the results of Section 6.3.

Table 6.1: Two Practical One-Way Hash Functions

Method	Hashing by the Rabin Encryption Function	Hashing by xDES
Complexity	Two Multiplications of Two 500-bit Integers (Modulo a 500-bit Integer)	Two Applications of xDES[1] (Six Applications of DES)
Efficiency	900-bit Strings $\Downarrow$ 500-bit Strings	224-bit Strings $\Downarrow$ 128-bit Strings
Index	$\alpha \in \Sigma^{50}$	$\alpha \in \Sigma^{56}$

Chapter 7

Structural Properties of One-Way Hash Functions

The main results of this chapter are as follows. First we prove that UOHs with respect to initial-strings chosen *arbitrarily* exist if and only if UOHs with respect to initial-strings chosen *uniformly at random* exist. Then, as an application of the result, we show that UOHs with respect to initial-strings chosen arbitrarily can be constructed under the assumption of the existence of one-way *quasi*-injections. Finally, we investigate relationships among various versions of one-way hash functions. We prove that some versions of one-way hash functions are strictly included in others by explicitly constructing hash functions that are one-way in the sense of the former but not in the sense of the latter.

7.1 Transforming UOH/U into UOH/$EN[\ell]$

Let P_1, P_2 be collections of ensembles with length $\ell(n)$. We say that UOH/P_1 is *transformable* into UOH/P_2 iff given a one-way hash function H in the sense of UOH/P_1, we can construct from H a one-way hash function H' in the sense of UOH/P_2. The main result of this section is Theorem 21 to be proved below, which states that UOH/U is transformable into UOH/$EN[\ell]$. Thus constructing one-way hash functions in the sense of UOH/$EN[\ell]$ under certain assumptions can be fulfilled in two steps: At the first step, we construct one-way hash functions in the sense of UOH/U. This would be easier, since a uniform ensemble would be easier to handle than arbitrary ones. Then at the second step, we apply the proof technique for Theorem 21 to obtain one-way hash functions in the sense of UOH/$EN[\ell]$.

To prove Theorem 21, we require a function family called *an invertible uniformizer*. Let T_n be a set of permutations over $\Sigma^{\ell(n)}$, and let $T = \bigcup_n T_n$. T is a

uniformizer with length $\ell(n)$ if it has the following properties 1, 2 and 3. Furthermore, F is *invertible* if it also has the following property 4.

1. For each n, for each pair of strings $x, y \in \Sigma^{\ell(n)}$, there are exactly $\sharp T_n / 2^{\ell(n)}$ permutations in T_n that map x to y.

2. There is a probabilistic polynomial time algorithm that on input n outputs a $t \in_R T_n$.

3. There is a polynomial time algorithm that computes all $t \in T$.

4. There is a polynomial time algorithm that computes t^{-1} for all $t \in T$.

The first property implies that for any $n \in \mathcal{N}$ and any $x \in \Sigma^{\ell(n)}$, when t is chosen randomly and uniformly from T_n, the probability that $t(x)$ coincides with a particular $y \in \Sigma^{\ell(n)}$ is

$$(\sharp T_n / 2^{\ell(n)}) / \sharp T_n = 1/2^{\ell(n)},$$

i.e., $t(x)$ is distributed randomly and uniformly over $\Sigma^{\ell(n)}$.

Now we give a concrete invertible uniformizer with length $\ell(n)$. Note that there is a natural one-to-one correspondence between strings of $\Sigma^{\ell(n)}$ and elements of $GF(2^{\ell(n)})$. So we will not distinguish $GF(2^{\ell(n)})$ from $\Sigma^{\ell(n)}$. Let a and b be elements of $GF(2^{\ell(n)})$ with $a \neq 0$. Then the affine transformation t defined by $t(x) = a \cdot x + b$ is a permutation over $GF(2^{\ell(n)})$, where $\cdot$ and $+$ are multiplication and addition over $GF(2^{\ell(n)})$ respectively. Denote by T_n the set of all the affine transformations on $GF(2^{\ell(n)})$ defined as above. Clearly, $\sharp T_n = 2^{\ell(n)}(2^{\ell(n)} - 1)$, and for any elements $x, y \in GF(2^{\ell(n)})$, there are exactly $(2^{\ell(n)} - 1) = \sharp T_n / 2^{\ell(n)}$ affine transformations in T_n that map x to y. In addition, generating $t \in_R T_n$ is easy, and for all $t \in T$, computing t and t^{-1} are simple tasks. Thus $T = \bigcup_n T_n$ is an invertible uniformizer with length $\ell(n)$. In Section 7.2, T will once again play a crucial role in constructing one-way hash functions in the sense of UOH/$EN[\ell]$ from one-way quasi-injections. Now we are ready to prove the following:

Theorem 21 *UOH/U is transformable into UOH/EN*$[\ell]$. [1]

Proof : Assume that H is a one-way hash function in the sense of UOH/U, where U is the uniform ensemble with length $\ell(n)$. We show how to construct from H a hash function H' that is one-way in the sense of UOH/$EN[\ell]$.

[1] In [DY90], De Santis and Yung obtained, independently, a restricted version of this theorem. They proved that *UOH/U is transformable into UOH/EN*[poly] by the use of the exactly same trick as ours.

Let $T = \bigcup_n T_n$ be an invertible uniformizer with length $\ell(n)$. Given H and $T = \bigcup_n T_n$, we construct H' as follows: $H' = \bigcup_n H'_n$, where

$$H'_n = \{h' \mid h' = h \circ t, h \in H_n, t \in T_n\}.$$

We claim that H' is one-way in the sense of UOH/$EN[\ell]$.

Assume for contradiction that H' is not one-way in the sense of UOH/$EN[\ell]$. Then there are a polynomial Q, an infinite subset $\mathcal{N}' \subseteq \mathcal{N}$, an ensemble E' with length $\ell(n)$ and a probabilistic polynomial time algorithm F' such that for all $n \in \mathcal{N}'$, the algorithm F', on input $x' \in_{E'} \Sigma^{\ell(n)}$ and $h' \in_R H'_n$, finds with probability $1/Q(n)$ a string $y' \in \Sigma^{\ell(n)}$ with $x' \neq y'$ and $h'(x') = h'(y')$. Now we show how to derive from F' a collision-string finder F that for all $n \in \mathcal{N}'$, on input $x \in_R \Sigma^{\ell(n)}$ and $h \in_R H_n$ where x is produced in a particular way to be described below, outputs with the same probability $1/Q(n)$ a string $y \in \Sigma^{\ell(n)}$ with $x \neq y$ and $h(x) = h(y)$.

Let M be a probabilistic Turing machine *with an oracle O that on input n outputs an $x' \in_{E'} \Sigma^{\ell(n)}$*. M produces $x \in_R \Sigma^{\ell(n)}$ in the following particular way:

1. Query the oracle O with n. Denote by x' the string answered by O. (Note that the oracle O is indispensable, as E' may be not samplable.)

2. Generate an $s \in_R T_n$ using its random tape.

3. Output $x = s(x')$.

From the first property of the uniformizer $T = \bigcup_n T_n$, we know that the ensemble E_M defined by the output of M is the uniform ensemble with length $\ell(n)$.

Let F be a probabilistic Turing machine. Note that F is allowed to receive from M information on how x is produced. In this case, the permutation s suffices. On input $x \in_{E_M} \Sigma^{\ell(n)}$ and $h \in_R H_n$, (important note: since E_M is the uniform ensemble with length $\ell(n)$, $x \in_{E_M} \Sigma^{\ell(n)}$ is equivalent to $x \in_R \Sigma^{\ell(n)}$), as well as s, F works as follows:

1. Calculate $z = s^{-1}(x)$. Note that $z = x' \in_{E'} \Sigma^{\ell(n)}$.

2. Call F' with input (z, h'), where $h' = h \circ t$. Note that $h' \in_R H'_n$, since $h \in_R H_n$ and $t \in_R T_n$.

3. Let $y' = F'(z, h')$. Output $y = y'$ whenever $y' =?$, and $y = t(y')$ otherwise.

Since F' is polynomial time bounded, F is also polynomial time bounded. Furthermore, since t is a permutation over $\Sigma^{\ell(n)}$, we have $y \neq ?$ (i.e. $x \neq y$ and $h(x) = h(y)$) iff $y' \neq ?$ (i.e. $x' \neq y'$ and $h'(x') = h'(y')$). Thus for all $n \in \mathcal{N}'$, F outputs, with the

same probability $1/Q(n)$, a string y such that $x \neq y$ and $h(x) = h(y)$, which implies that H is *not* a one-way hash function in the sense of UOH/U, a contradiction.

From the above discussions we know that H' is indeed a one-way hash function in the sense of UOH/$EN[\ell]$. This completes the proof. □

A significant corollary of Theorem 21 is:

Corollary 1 *One-way hash functions in the sense of UOH/$EN[\ell]$ exist iff those in the sense of UOH/U exist.*

7.2 UOHs from One-way Quasi-Injections

As an application of Theorem 21, in this section we construct one-way hash functions in the sense of UOH/$EN[\ell]$ under the assumption of the existence of one-way *quasi*-injections. Main ingredients of our construction include

1. one-way quasi-injections,
2. universal hash functions with the correspondence accessibility property,
3. pair-wise independent uniformizers and,
4. invertible uniformizers.

Our construction is partially inspired by [NY89].

7.2.1 One-Way Quasi-Injections

Assume that f is a one-way function from $\bigcup_n \Sigma^n$ to $\bigcup_n \Sigma^{\ell(n)}$. A string $x \in \Sigma^n$ is said to *have a sibling* if there is a string $y \in \Sigma^n$ such that $f_n(x) = f_n(y)$.

Definition 12 *A one-way function f is a one-way* quasi-*injection iff for any polynomial Q and for all sufficiently large $n \in \mathcal{N}$,*

$$\sharp SIB_n/2^n < 1/Q(n)$$

where SIB_n is the collection of all strings in Σ^n that have siblings.

Let V_n be a set of permutations over $\Sigma^{\ell(n)}$, and $V = \bigcup_n V_n$. V is a *pair-wise independent uniformizer* with length $\ell(n)$ if it has the following three properties.

1. For each n, for any pairs of strings (x_1, x_2) and (y_1, y_2), there are exactly $\sharp V_n/[2^{\ell(n)}(2^{\ell(n)}-1)]$ permutations in V_n that map x_1 to y_1 and x_2 to y_2, where $x_1, x_2, y_1, y_2 \in \Sigma^{\ell(n)}$, $x_1 \neq x_2$, $y_1 \neq y_2$, and $2^{\ell(n)}(2^{\ell(n)}-1)$ is the total number of ordered pairs (x, y) with $x \neq y$ and $x, y \in \Sigma^{\ell(n)}$.

2. There is a probabilistic polynomial time algorithm that on input n outputs a $v \in_R V_n$.

3. There is a polynomial time algorithm that computes all $v \in V$.

Similar to uniformizers defined in Section 7.1, the first property implies that for any $n \in \mathcal{N}$ and any (x_1, x_2) with $x_1 \neq x_2$ and $x_1, x_2 \in \Sigma^{\ell(n)}$, when v is chosen randomly and uniformly from V_n, $(v(x_1), v(x_2))$ is distributed randomly and uniformly over all ordered pairs (y_1, y_2) with $y_1 \neq y_2$ and $y_1, y_2 \in \Sigma^{\ell(n)}$.

Recall the invertible uniformizer $T = \bigcup_n T_n$ constructed in Section 7.1. For any $x_1, x_2, y_1, y_2 \in \Sigma^{\ell(n)}$ with $x_1 \neq x_2$ and $y_1 \neq y_2$, there is exactly one permutation in T_n that maps x_1 to y_1 and x_2 to y_2. Note that

$$1 = 2^{\ell(n)}(2^{\ell(n)}-1)/2^{\ell(n)}(2^{\ell(n)}-1) = \sharp T_n/[2^{\ell(n)}(2^{\ell(n)}-1)],$$

which implies that T is a pair-wise independent uniformizer.

7.2.2 The Construction

Assume that we are given a one-way quasi-injection f from D to R where $D = \bigcup_n \Sigma^n$, $R = \bigcup_n \Sigma^{m(n)}$ and m is a polynomial with $m(n) \geq n$. Let $V = \bigcup_n V_n$ be a pair-wise independent uniformizer with length $m(n)$, and $S = \bigcup_n S_n$ be a strong universal$_2$ hash function that compresses $m(n)$-bit input into $(n-1)$-bit output strings and has the correspondence accessibility property.

Lemma 9 *Let $H_n = \{h \mid h = s \circ v \circ f_{n+1}, s \in S_{n+1}, v \in V_{n+1}\}$, and $H = \bigcup_n H_n$. Then H is a one-way hash function in the sense of UOH/U compressing $(n+1)$-bit input into n-bit output strings, under the assumption that f is a one-way quasi-injection.*

Proof : Assume for contradiction that H is not one-way in the sense of UOH/U. Then there are a polynomial Q_1, an infinite subset $\mathcal{N}' \subseteq \mathcal{N}$ and a collision-string finder F such that for all $n \in \mathcal{N}'$, the finder F, on input $x \in_R \Sigma^{n+1}$ and $h \in_R H_n$, outputs with probability at least $1/Q_1(n)$ a string $y \in \Sigma^{n+1}$ with $x \neq y$ and $h(x) = h(y)$. We show that F can be used to construct an algorithm M that for all sufficiently large $n \in \mathcal{N}'$, inverts f_{n+1} with probability greater than $1/2Q_1(n)$.

Assume that $w \in_R \Sigma^{n+1}$ and $z = f_{n+1}(w)$. On input z, the algorithm M runs as follows in trying to compute a y such that $z = f_{n+1}(y)$:

Algorithm M:

1. Generate an $x \in_R \Sigma^{n+1}$. If $z = f_{n+1}(x)$ then output $y = x$ and halt. Otherwise execute the following steps.
2. Generate a $v \in_R V_{n+1}$.
3. Let $u_1 = v \circ f_{n+1}(x)$ and $u_2 = v(z)$. Choose a random $s \in S_{n+1}$ such that $s(u_1) = s(u_2)$. This is possible according to the correspondence accessibility property of S.
4. Let $h = s \circ v \circ f_{n+1}$. Call F with input h and x, and output $y = F(x, h)$.

First we show that h produced by M is a random element in H_n. At Step 2, a $v \in_R V_{n+1}$ is generated. Since $f_{n+1}(x) \neq z$, from the first property of V we know that $(v \circ f_{n+1}(x), v(z))$ is distributed randomly and uniformly over all pairs (x_1, x_2) with $x_1 \neq x_2$ and $x_1, x_2 \in \Sigma^{m(n+1)}$. At Step 3, s is chosen uniformly at random from all those functions in S_{n+1} that map u_1 and u_2 to the same string. Consequently, $h = s \circ v \circ f_{n+1}$ is a random element in H_n.

The running time of M is clearly polynomial in n. Next we estimate the probability that M outputs y such that $z = f_{n+1}(y)$. Denote by $\mathrm{Inv}(z)$ the set $\{e \mid z = f_{n+1}(e), e \in \Sigma^{n+1}\}$. Then M halts at Step 1 iff $x \in \mathrm{Inv}(z)$.

First we note that

$$\Pr\{z = f_{n+1}(y)\} \geq \Pr\{x \in \Sigma^{n+1} - \mathrm{Inv}(z), x \text{ has no sibling}, z = f_{n+1}(y)\},$$

where $\Pr\{z = f_{n+1}(y)\}$ is computed over Σ^{n+1}, Σ^{n+1}, V_{n+1}, S_{n+1} and the sample space of all finite strings of coin flips that F could have tossed. Note that the two compound events "$x \in \Sigma^{n+1} - \mathrm{Inv}(z)$, x has no sibling, $z = f_{n+1}(y)$" and "$x \in \Sigma^{n+1} - \mathrm{Inv}(z)$, x has no sibling, $y \neq ?$" are in fact the same. So the probability $\Pr\{z = f_{n+1}(y)\}$ can be estimated via the probability $\Pr\{x \in \Sigma^{n+1} - \mathrm{Inv}(z), x \text{ has no sibling}, y \neq ?\}$. Now we focus on the latter. By assumption, we have $\Pr\{y \neq ?\} \geq 1/Q_1(n)$ for all $n \in \mathcal{N}'$, where $\Pr\{y \neq ?\}$ is computed over Σ^{n+1}, V_{n+1}, S_{n+1} and the sample space of all finite strings of coin flips that F could have tossed. On the other hand,

$$\begin{aligned}
\Pr\{y \neq ?\} &= \Pr\{x \in \mathrm{Inv}(z), y \neq ?\} + \Pr\{x \in \Sigma^{n+1} - \mathrm{Inv}(z), y \neq ?\} \\
&= \Pr\{x \in \mathrm{Inv}(z), y \neq ?\} + \\
&\quad \Pr\{x \in \Sigma^{n+1} - \mathrm{Inv}(z), x \text{ has a sibling}, y \neq ?\} + \\
&\quad \Pr\{x \in \Sigma^{n+1} - \mathrm{Inv}(z), x \text{ has no sibling}, y \neq ?\}.
\end{aligned}$$

Recall that f is one-way. So for all sufficiently large $n \in \mathcal{N}$, we have

$$\Pr\{x \in \mathrm{Inv}(z), y \neq ?\} \leq \Pr\{x \in \mathrm{Inv}(z)\} < 1/4Q_1(n).$$

Furthermore, for all sufficiently n we have

$$\Pr\{x \in \Sigma^{n+1} - \mathrm{Inv}(z), x \text{ has a sibling}, y \neq ?\} \leq \Pr\{x \text{ has a sibling}\} < 1/4Q_1(n),$$

since f is a one-way quasi-injection. Thus for all sufficiently large $n \in \mathcal{N}'$,

$$\begin{aligned}
\Pr\{z = f_{n+1}(y)\} &\geq \Pr\{x \in \Sigma^{n+1} - \mathrm{Inv}(z), x \text{ has no sibling}, z = f_{n+1}(y)\} \\
&= \Pr\{x \in \Sigma^{n+1} - \mathrm{Inv}(z), x \text{ has no sibling}, y \neq ?\} \\
&\geq 1/Q_1(n) - [\Pr\{x \in \mathrm{Inv}(z), y \neq ?\} + \\
&\quad \Pr\{x \in \Sigma^{n+1} - \mathrm{Inv}(z), x \text{ has a sibling}, y \neq ?\}] \\
&\geq 1/Q_1(n) - [1/4Q_1(n) + 1/4Q_1(n)] \\
&\geq 1/2Q_1(n).
\end{aligned}$$

This contradicts our assumption that f is a one-way quasi-injection, and hence the theorem follows. □

Combining Theorem 21 and Lemma 9, we have the following result: *A one-way hash function H' in the sense of UOH/$EN[\ell']$, where ℓ' is defined by $\ell'(n) = n + 1$, can be constructed under the assumption that f is a one-way quasi-injection.* By an argument analogous to that of Theorem 3.1 of [Dam89], it can be proved that for any polynomial ℓ, we can construct from H' a one-way hash function H'' in the sense of UOH/$EN[\ell]$. Thus:

Theorem 22 *One-way hash functions in the sense of UOH/$EN[\ell]$ can be constructed assuming the existence of one-way quasi-injections.*

Similarly, we can construct one-way hash functions in the sense of $\mathrm{UOH}_C/EN[\ell]$ assuming the existence of one-way quasi-injections *with respect to the circuit model.*

7.3 A Hierarchy of One-Way Hash Functions

In this section, we discuss relationships among various versions of one-way hash functions: UOH/U, UOH/$PSE[\ell]$, UOH/$EN[\ell]$, $\mathrm{UOH}_C/EN[\ell]$, UOH/EN[poly], CIH, CIH_C and CIH/EN[poly].

First we define a relation between two versions, Ver_1 and Ver_2, of one-way hash functions. We say that

1. Ver_1 is *included in* Ver_2, denoted by $Ver_1 \subseteq Ver_2$, if all one-way hash functions in the sense of Ver_1 are also one-way hash functions in the sense of Ver_2.

2. Ver_1 is *strictly* included in Ver_2, denoted by $Ver_1 \subset Ver_2$, if $Ver_1 \subseteq Ver_2$ and there is a one-way hash function in the sense of Ver_2 but not in the sense of Ver_1.

3. Ver_1 and Ver_2 are *equivalent*, denoted by $Ver_1 = Ver_2$, if $Ver_1 \subseteq Ver_2$ and $Ver_2 \subseteq Ver_1$.

Lemma 10 *The following statements hold:*

(1) $CIH_C = CIH/EN[\text{poly}]$.

(2) $UOH_C/EN[\ell] = UOH/EN[\text{poly}]$.

(3) $UOH/EN[\text{poly}] \subseteq UOH/EN[\ell] \subseteq UOH/PSE[\ell] \subseteq UOH/U$.

(4) $CIH/EN[\text{poly}] \subseteq CIH$.

(5) $CIH \subseteq UOH/PSE[\ell]$.

(6) $CIH/EN[\text{poly}] \subseteq UOH/EN[\text{poly}]$.

Proof : Proofs for (1) and (2) are analogous to that for "polynomial size circuits vs. P/poly" [Pip79]. (3),(4), (5) and (6) are obvious. Here we give a detailed description for the proof of (1). Proof for (2) is similar, and is omitted.

The "$\subseteq$" part: Assume that H is a one-way hash function in the sense of CIH_C. If H is not one-way in the sense of $\text{CIH}/EN[\text{poly}]$, then there are polynomials Q_1 and Q_2, an infinite subset $\mathcal{N}' \subseteq \mathcal{N}$, an ensemble E with length $Q_2(n)$, and a collision-pair finder F, such that for all $n \in \mathcal{N}'$, the finder F, on input $z \in_E \Sigma^{Q_2(n)}$ and $h \in_R H_n$, outputs a collision-pair with probability $1/Q_1(n)$. Note that for each $n \in \mathcal{N}$ and $h \in_R H_n$, the probability that F successfully outputs a collision-pair is computed over $\Sigma^{Q_2(n)}$ and the sample space of all finite strings of coin flips that F could have tossed. Let $z_{\max}$ be the first string according to the lexicographic order in $\Sigma^{Q_2(n)}$ such that for $h \in_R H_n$, F outputs a collision-pair with the maximum probability, which is certainly at least $1/Q_1(n)$. F can be converted into a family $A = \{A_n \mid n \in \mathcal{N}\}$ of probabilistic polynomial size circuits with $z_{\max}$ being "embedded in" A_n. Thus for each $n \in \mathcal{N}'$, A_n on input $h \in_R H_n$ outputs a collision-pair with probability at least $1/Q_1(n)$. In other words, H is not one-way in the sense of CIH_C, which is a contradiction.

The "$\supseteq$" part: Assume that H is a one-way hash function in the sense of $\text{CIH}/EN[\text{poly}]$. If H is not one-way in the sense of CIH_C, then there are a polynomial Q_1, an infinite

subset $\mathcal{N}' \subseteq \mathcal{N}$, and a collision-pair finder $A = \{A_n \mid n \in \mathcal{N}\}$, such that for all $n \in \mathcal{N}'$, A_n outputs a collision-pair with probability $1/Q_1(n)$. Since the size of A is polynomially bounded, there is a polynomial Q_2 such that the description of A_n is not longer than $Q_2(n)$ for all $n \in \mathcal{N}$. Without loss of generality, assume that the description of A_n is exactly $Q_2(n)$ bits long. Let E be the ensemble with length $Q_2(n)$ defined by $E_n(x) = 1$ whenever x is the description of A_n, and $E_n(x) = 0$ otherwise. Note that E may be not samplable.

Recall that the (probabilistic) *circuit value problem* is (probabilistic) polynomial time computable (see [BDG88], p.110). So there is a (probabilistic) polynomial time algorithm F that on input $z \in_E \Sigma^{Q_2(n)}$ and $h \in_R H_n$, (Note: By the definition of E, we have z=the description of A_n), output a collision-pair with probability $1/Q(n)$. This implies that H is not one-way in the sense of CIH/EN[poly], which contradicts our assumption. □

Theorem 23 *The following statements hold:*

(1) $UOH/PSE[\ell] \subset UOH/U$.

(2) *There are one-way hash functions in the sense of* UOH/EN[poly] *but not in the sense of CIH.*

(3) $CIH \subset UOH/PSE[\ell]$.

(4) CIH/EN[poly] $\subset UOH/EN$[poly].

Proof: (1) We show that given a one-way hash function H in the sense of UOH/U, we can construct from H a hash function H' that is still one-way in the sense of UOH/U but not in the sense of UOH/$PSE[\ell]$.

H' is constructed as follows: Denote by $0^{\ell(n)}$ ($1^{\ell(n)}$, respectively) the all-0 (all-1, respectively) string of length $\ell(n)$. For each $h \in H_n$, define a function $h' : \Sigma^{\ell(n)} \to \Sigma^n$ by $h'(x) = h(0^{\ell(n)})$ whenever $x = 1^{\ell(n)}$ and $h'(x) = h(x)$ otherwise. See also Figure 7.1. Thus the only difference between h and h' is the images of $1^{\ell(n)}$. Let H'_n be the collection of all h', and let $H' = \bigcup_n H'_n$. We claim that H' is still one-way in the sense of UOH/U but not in the sense of UOH/$PSE[\ell]$.

Let M be a polynomial time algorithm that on input n outputs $1^{\ell(n)}$. By definition, the ensemble E defined by the output of M is polynomially samplable. Let F be a collision-string finder that on input x and h' outputs the string $0^{\ell(n)}$ whenever $x = 1^{\ell(n)}$ and "?" otherwise. Clearly, for all n, $x \in_E \Sigma^{\ell(n)}$ and $h' \in H'_n$, F always finds a string y that collides with x. Therefore H' is not one-way in the sense of UOH/$PSE[\ell]$.

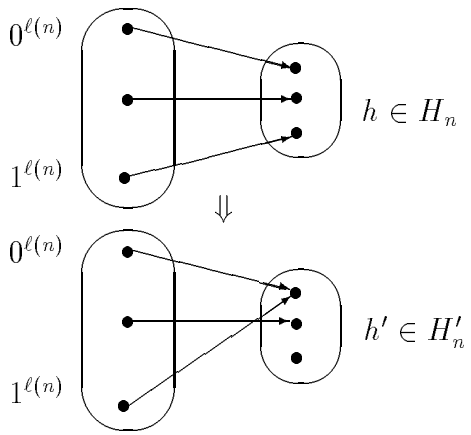


Figure 7.1: UOH/$PSE[\ell]$ is Strictly Included in UOH/U

Now we prove that H' is one-way in the sense of UOH/U. Assume for contradiction that H' is not one-way in the sense of UOH/U. Then there are an infinite subset $\mathcal{N}' \subseteq \mathcal{N}$ and a collision-string finder F such that for some polynomial Q and for all $n \in \mathcal{N}'$, $\Pr\{F(x,h') \neq ?\} \geq 1/Q(n)$, when $x \in_R \Sigma^{\ell(n)}$ and $h' \in_R H'_n$.

Note that

$$\begin{aligned}
&\Pr\{F(x,h') \neq ?\} \\
&= \Pr\{F(x,h') \neq ? \mid h'(x) = h'(0^{\ell(n)})\} \cdot \Pr\{h'(x) = h'(0^{\ell(n)})\} + \\
&\quad \Pr\{F(x,h') \neq ? \mid h'(x) \neq h'(0^{\ell(n)})\} \cdot \Pr\{h'(x) \neq h'(0^{\ell(n)})\} \\
&\geq 1/Q(n),
\end{aligned}$$

and that

$$\begin{aligned}
&\Pr\{F(x,h') \neq ? \mid h'(x) = h'(0^{\ell(n)})\} \cdot \Pr\{h'(x) = h'(0^{\ell(n)})\} \\
&\leq \Pr\{h'(x) = h'(0^{\ell(n)})\} \\
&\leq \Pr\{h(x) = h(0^{\ell(n)})\} + 1/2^{\ell(n)} \\
&\leq 2\Pr\{h(x) = h(0^{\ell(n)})\}.
\end{aligned}$$

Since H is one-way in the sense of UOH/U, we have

$$\Pr\{h(x) = h(0^{\ell(n)})\} < 1/4Q(n)$$

for all sufficiently large n. Thus for all sufficiently large $n \in \mathcal{N}'$,

$$\Pr\{F(x,h') \neq ? \mid h'(x) \neq h'(0^{\ell(n)})\}$$

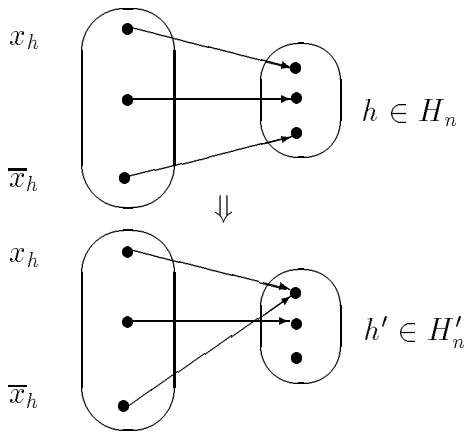


Figure 7.2: UOH/PSE[poly] versus CIH

$$\begin{aligned} &\geq \Pr\{F(x,h') \neq ? \mid h'(x) \neq h'(0^{\ell(n)})\} \cdot \Pr\{h'(x) \neq h'(0^{\ell(n)})\} \\ &\geq 1/Q(n) - \Pr\{F(x,h') \neq ? \mid h'(x) = h'(0^{\ell(n)})\} \cdot \Pr\{h'(x) = h'(0^{\ell(n)})\} \\ &> 1/2Q(n). \end{aligned}$$

By definition, when $h'(x) \neq h'(0^{\ell(n)})$, a string $y \in \Sigma^{\ell(n)}$ with $x \neq y$ collides with x *under* h' iff it does *under* h. Consequently, the collision-string finder F can be used to "break" H, this implies that H is not one-way in the sense of UOH/U, a contradiction.

(2) The proof is very similar to that for (1). Given H, a one-way hash function in the sense of UOH/EN[poly], we construct a hash function H' that is still one-way in the sense of UOH/EN[poly] but not in the sense of CIH.

Without loss of generality, assume that the length of the description of $h \in H_n$ is greater than $n/2$, and for any distinct $h_1, h_2 \in H_n$ the first $n/2$ bits of h_1 is different from that of h_2. For each $h \in H_n$, we associate with it a particular $\ell(n)$-bit string x_h that is obtained by repeatedly concatenating the first $n/2$ bits of the description of h until the length of the resulting string becomes $\ell(n)$.

For each $h \in H_n$, define a function $h' : \Sigma^{\ell(n)} \to \Sigma^n$ by $h'(x) = h(x_h)$ whenever $x = \overline{x}_h$ and $h'(x) = h(x)$ otherwise, where $\overline{x}_h$ is the complement of x_h. See also Figure 7.2. Thus the only difference between h and h' is the images of $\overline{x}_h$. Let H'_n be the collection of all h', and let $H' = \bigcup_n H'_n$. By analyses similar to (1), one can verify that H' is still one-way in the sense of UOH/EN[poly] but not in the sense of CIH.

(3) follows from (2) and $CIH \subseteq$ UOH/$PSE[\ell]$. (4) follows from (2) and the facts

that CIH/EN[poly] $\subseteq CIH$ and that CIH/EN[poly] $\subseteq$ UOH/EN[poly]. $\square$

From Lemma 10 and Theorem 23, we have the following hierarchical structure for one-way hash functions (see Figure 7.3).

$$
\begin{array}{ccc}
 & & \mathrm{UOH}/U \\
 & & \cup \\
\mathrm{CIH} & \subset & \mathrm{UOH}/PSE[\ell] \\
 & & \cup \\
\cup & & \mathrm{UOH}/EN[\ell] \\
 & & \cup \\
\mathrm{CIH}/EN[\mathrm{poly}] & \subset & \mathrm{UOH}/EN[\mathrm{poly}] \\
| \; | & & | \; | \\
\mathrm{CIH}_C & & \mathrm{UOH}_C/EN[\ell]
\end{array}
$$

Figure 7.3: Hierarchical Structure of One-Way Hash Functions

Recently, substantial progress on the *construction* of UOHs has been made by De Santis and Yung [DY90], and especially, by Rompel [Rom90] who finally solved the problem of constructing UOHs under the sole assumption of the existence of one-way functions.

Chapter 8

Conclusion

We have investigated issues in designing secure secret-key block ciphers and one-way hash functions. In Part I, we have proposed various types of transformations, and showed that among them Generalized Type-2 transformations are the most preferable. Two provably secure block ciphers have been constructed by the use of Generalized Type-2 transformations, and based on the latter one of the two ciphers, practically implementable block ciphers have been presented.

In Part II, we have revealed a duality between constructions of pseudo-random string generators and one-way hash functions. Applying the duality, we have presented a simple construction for universal one-way hash functions assuming the existence of one-way permutations. Using ideas behind the construction, we have proposed practical one-way hash functions, the fastest of which compress nearly $2n$-bit long input into n-bit long output strings by applying only twice a one-way function.

We have also proved that UOHs with respect to initial-strings chosen uniformly at random can be transformed into UOHs with respect to initial-strings chosen arbitrarily, and that UOHs with respect to initial-strings chosen arbitrarily can be constructed assuming the existence of one-way quasi-injections. We have also investigated relationships among various versions of one-way hash functions. In particular, we have shown that UOH/$PSE[\ell]$, CIH and CIH/EN[poly] are strictly included in UOH/U, UOH/$PSE[\ell]$ and UOH/EN[poly] respectively, and that there are one-way hash functions in the sense of UOH/EN[poly] but not in the sense of CIH.

Finally, we end the thesis with several open problems as well as directions for future research.

1. In Chapter 2, the main impossibility result on the Feistel-type transformation has been generalized the Type-1 transformations. It is interesting to generalize further the result to other, such as Type-2, transformations.

2. Can the results in Chapters 3 – 4 be made finer ?

3. Can the efficiency of the universal one-way hash function H constructed in Section 6.2.2 be improved by a factor larger than $O(\log n)$? Is H also a CIH ?

4. By Theorem 23 in Chapter 7, there are one-way hash functions in the sense of UOH/EN[poly] but not in the sense of CIH. However, it is not clear whether or not CIH is *strictly* included in UOH/EN[poly], as we do not know whether or not CIH $\subseteq$ UOH/EN[poly].

Bibliography

[BDG88] J. Balcázar, J. Díaz and J. Gabarró: *Structural Complexity I*, EATCS Monographs on Theoretical Computer Science, Springer-Verlag, Berlin, 1988.

[BM84] M. Blum and S. Micali: "How to generate cryptographically strong sequences of pseudo-random bits", *SIAM Journal on Computing*, Vol.13, No.4, 1984, pp.850-864.

[BH89] R. Boppana and R. Hirschfeld: "Pseudorandom generations and complexity classes", *Randomness and Computation*, Advances in Computing Research, Vol.5, JAI Press Inc., 1989, pp.1-26.

[CW79] J. Carter and M. Wegman: "Universal classes of hash functions", *Journal of Computer and System Sciences*, Vol.18, 1979, pp.143-154.

[Dam87] I. Damgård: "Collision free hash functions and public key signature schemes", *Advances in Cryptology — EuroCrypt'87*, Lecture Notes in Computer Science, Vol.304, Springer-Verlag, 1988, pp.203-216.

[Dam89] I. Damgård: "A design principle for hash functions", *Advances in Cryptology — Crypto'89*, Lecture Notes in Computer Science, Vol.435, Springer-Verlag, 1990, pp.416-427.

[DY90] A. De Santis and M. Yung: "On the design of provably-secure cryptographic hash functions", To appear in *Advances in Cryptology — EuroCrypt'90*, Lecture Notes in Computer Science, Springer-Verlag.

[FNS75] H. Feistel, W.A. Notz and J.L. Smith: "Some cryptographic techniques for machine-to-machine data communications", *Proceedings of IEEE*, Vol.63, No. 11, 1975, pp.1545-1554.

[GGM86] O. Goldreich, S. Goldwasser and S. Micali: "How to construct random functions", *Journal of ACM*, Vol.33, No.4, 1986, pp.792-807.

[GM84] S. Goldwasser and S. Micali: "Probabilistic encryption", *Journal of Computer and System Sciences*, Vol.28, 1984, pp.270-299.

[Has90] J. Håstad: "Pseudo-random generation under uniform assumptions", *Proceedings of the 22-nd ACM Symposium on Theory of Computing*, 1990, pp.395-404.

[ILL89] R. Impagliazzo, L. Levin and M. Luby: "Pseudo-random generation from one-way functions", *Proceedings of the 21-st ACM Symposium on Theory of Computing*, 1989, pp.12-24.

[IL89] R. Impagliazzo and M. Luby: "One-way functions are essential for complexity based cryptography", *Proceedings of the 30-th IEEE Symposium on the Foundations of Computer Science*, 1989, pp.230-235.

[KL82] R. Karp and R. Lipton: "Turing machines that take advice", *L'enseigment Mathematique*, Vol.28, 1982, pp.191-209.

[Kon81] A.G. Konheim: *Cryptography: A Primer*, John Wiley & Sons, Inc. 1981.

[Leo90] J. Leo: "Definitions of one-way hash functions", *Manuscript*, August 1990.

[Lev87] L.A. Levin: "One-way functions and pseudorandom generators", *Combinatorica*, Vol.7, No.4, 1987, pp.357-363.

[LR88] M. Luby and C. Rackoff: "How to construct pseudorandom permutations from pseudorandom functions", *SIAM Journal on Computing*, Vol.17, No.2, 1988, pp.373-386. (A preliminary version including other results appeared in *Proceedings of the 18th ACM Symposium on Theory of Computing*, 1986, pp.356-363.)

[MM89] U.M. Maurer and J.L. Massey: "Perfect local randomness in pseudo-random sequences", *Advances in Cryptology — Crypto'89*, Lecture Notes in Computer Science, Vol.435, Springer-Verlag, 1990, pp.100-112.

[Mer89] R. Merkle: "One way hash functions and DES", *Advances in Cryptology — Crypto'89*, Lecture Notes in Computer Science, Vol.435, Springer-Verlag, 1990, pp.428-446.

[Mey78] C.H. Meyer: "Ciphertext/plaintext and ciphertext/key dependence vs number of rounds for the data encryption standard", *AFIPS Conference Proceedings*, Vol.47, 1978, pp.1119-1126.

[MSch88] S. Micali and C.P. Schnorr: "Super-efficient, perfect random number generators", *Advances in Cryptology — Crypto'88*, Lecture Notes in Computer Science, Vol.403, Springer-Verlag, 1990, pp.173-198.

[NBS77] *Data Encryption Standard*, Federal Information Processing Standards (FIPS) Publication 46, National Bureau of Standards, U.S. Department of Commerce, 1977.

[NY89] M. Naor and M. Yung: "Universal one-way hash functions and their cryptographic applications", *Proceedings of the 21-st ACM Symposium on Theory of Computing*, 1989, pp.33-43.

[NS90] K. Nishimura and M. Sibuya: "Probability to meet in the middle", *Journal of Cryptology*, Vol.2, No.1, 1990, pp.13-22.

[Oh88] Y. Ohnishi: "A study on data security", *Master Thesis* (in Japanese), Tohoku University, Japan, March 1988.

[Pie90] J. Pieprzyk: "How to construct pseudorandom permutations from single pseudorandom functions", To appear in *Advances in Cryptology — EuroCrypt'90*, Lecture Notes in Computer Science, Springer-Verlag.

[Pip79] N. Pippenger: "On simultaneous resource bounds", *Proceedings of the 20-th IEEE Symposium on the Foundations of Computer Science*, 1979, pp.307-311.

[Rom90] J. Rompel: "One-way functions are necessary and sufficient for secure signatures", *Proceedings of the 22-nd ACM Symposium on Theory of Computing*, 1990, pp.387-394.

[Rue89] R.A. Rueppel: "On the security of Schnorr's pseudorandom generator", *Advances in Cryptology — EuroCrypt'89*, Lecture Notes in Computer Science, Vol.434, Springer-Verlag, 1990, pp.423-428.

[Sch88] C.P. Schnorr: "On the construction of random number generators and random function generators", *Advances in Cryptology — EuroCrypt'88*, Lecture Notes in Computer Science, Vol.330, Springer-Verlag, 1988, pp.225-232.

[Wa88] O. Watanabe: "On one-way functions", Presented at *the International Symposium on Combinatorial Optimization*, Tianjin, China, August 1988.

[WC81] M. Wegman and J. Carter: "New hash functions and their use in authentication and set equality", *Journal of Computer and System Sciences*, Vol.22, 1981, pp.265-279.

[Yao82] A. Yao: "Theory and applications of trapdoor functions", *Proceedings of the 23-rd IEEE Symposium on the Foundations of Computer Science*, 1982, pp.80-91.

[ZMI89a] Y. Zheng, T. Matsumoto and H. Imai: "Impossibility and optimality results on constructing pseudorandom permutations", *Advances in Cryptology — EuroCrypt'89*, Lecture Notes in Computer Science, Vol.434, Springer-Verlag, 1990, pp.412-422.

[ZMI89b] Y. Zheng, T. Matsumoto and H. Imai: "On the construction of block ciphers provably secure and not relying on any unproved hypotheses", *Advances in Cryptology — Crypto'89*, Lecture Notes in Computer Science, Vol.435, Springer-Verlag, 1990, pp.461-480.

[ZMI90a] Y. Zheng, T. Matsumoto and H. Imai: "Duality between two cryptographic primitives", To appear in *Proceedings of the 8-th International Conference on Applied Algebra, Algebraic Algorithms and Error Correcting Codes (AAECC-8)*, Lecture Notes in Computer Science, Springer-Verlag. A preliminary version appeared as *IEICE Technical Report on Information Security*, ISEC89-46, Japan, March 1990.

[ZMI90b] Y. Zheng, T. Matsumoto and H. Imai: "Connections among several versions of one-way hash functions", *Special Issue on Cryptography and Information Security, Transactions of IEICE, Japan*, Vol.**E**73, No.7, pp.1092-1099, July 1990. A preliminary version appeared in *Proceedings of the 1990 Symposium on Cryptography and Information Security (SCIS90)*, Nihondaira, Japan, January 1990.

[ZMI90c] Y. Zheng, T. Matsumoto and H. Imai: "Structural properties of one-way hash functions", To appear in *Advances in Cryptology — Crypto'90*, Lecture Notes in Computer Science, Springer-Verlag.

Vita

Yuliang Zheng was born in Jiangsu, China on February 5, 1962. He received the B.S. degree in computer science from the Southeastern University (formerly the Nanjing Institute of Technology), Nanjing, China, in 1982, and the M.E. degree in electrical and computer engineering from the Yokohama National University, Yokohama, Japan, in 1988. From 1982 to 1984, he was a research staff member of the Guangzhou Research Institute for Communications, Guangzhou, China.