Chapter 2
Some Useful Constructions

Almost every protocol described in this book takes advantage of some, or all of the following three basic strategies of utilizing a PRF \( h() \): (a) hash chains; (b) hash trees, which are also referred to as binary hash chains; and (c) the uniqueness of random subsets of large sets generated using PRF \( h() \).

2.1 Hash Chains

A hash chain is [9] constructed through successive applications of the PRF \( h() \) on a bit-string \( X_0 \).

For example, let

\[
X_1 = h(X_0), X_2 = h(X_1), X_3 = h(X_2) \ldots, X_n = h(X_{n-1}).
\]  

(2.1)

Such \( n \) successive applications that result in the value \( X_n \) can be conveniently represented by the notation

\[
X_n = h^n(X_0).
\]  

(2.2)

From the properties of a PRF \( h() \) it follows that given a value \( X_i \) from a hash chain, it is

1. Easy to compute \( X_j \), if \( j \geq i \), through \( j - i \) applications of \( h() \)
2. Infeasible to compute \( X_j \), if \( j < i \)

Furthermore, given two values \( U \) and \( V \) satisfying \( h^i(U) = V \), even while there exists numerous values \( U' \) satisfying \( h^i(U') = V \), it is safe to conclude that \( V \) was indeed generated by repeatedly hashing \( U \).

Even while the input and output to \( h() \) appear to have the same size (\( u \)-bits), it should be assumed that the input is padded with \( l \) fixed pad-bits to size \( l + u \).
2.1.1 Hash Accumulator

Given a list of values \( v_1 \cdots v_n \), a hash accumulator computes an accumulated hash \( \alpha \) as follows.

\[
\begin{align*}
\alpha_2 &= h(v_1 \parallel v_2) \\
\alpha_3 &= h(\alpha_2 \parallel v_3) \\
\alpha_4 &= h(\alpha_3 \parallel v_4) \\
& \vdots \\
\alpha &= h(\alpha_{n-1} \parallel v_n)
\end{align*}
\] (2.3)

Each step in the accumulation of the hash is also referred to as hash-extension. For example, in the operation \( h(\alpha_2 \parallel v_3) \), “\( \alpha_2 \) is hash-extended with \( v_3 \).”

The accumulated hash can be seen as a commitment to all values \( v_1 \cdots v_n \). Specifically, even while there are numerous possible sets of values which yield the same accumulated value \( \alpha \), given \( \alpha \) and the values \( v_1 \cdots v_n \), one can conclude that \( \alpha \) was indeed computed by accumulating values \( v_1 \cdots v_n \).

2.1.2 Hash Tree

A more common strategy for accumulating a set of values \( v_1 \cdots v_n \) into a single commitment \( \alpha \) is by arranging values \( v_1 \cdots v_n \) as leaves of a binary hash tree. The binary hash tree is more commonly referred to as a Merkle tree [10]. For simplicity, we shall assume that \( n \) is a power of 2.

Figure 2.1 depicts a Merkle tree with \( N = 16 \) leaves \( v_0 \cdots v_f \). A binary tree with \( N \) leaf-nodes has \( 2N - 1 \) nodes spread over \( \log_2 N + 1 \) levels—levels \( 0 \cdots L = \log_2 N \).
At level 0 are the \( N \) leaf-nodes \( v_0 \cdots v_f \). At level 1 are \( N/2 \) nodes, each obtained by hashing together two adjacent nodes in level 0. In the figure, the eight nodes \( v_{01}, v_{23}, \ldots v_{ef} \) in level 1 are obtained as

\[
\begin{align*}
v_{01} &= h(v_0 \parallel v_1) \\
v_{23} &= h(v_2 \parallel v_3) \\
& \vdots \\
v_{ef} &= h(v_e \parallel v_f)
\end{align*}
\]  

(2.4)

Similarly, the four nodes at level 2 are each obtained by hashing together two adjacent nodes in level 1. For example,

\[
v_{03} = h(v_{01} \parallel v_{23}).
\]  

(2.5)

Note that a tree with \( N = 2^L \) leaves at level 0 has \( 2^{L-i} \) nodes in level \( i \), where \( i = 0 \cdots L \). The total number of nodes in the tree is thus

\[
\sum_{i=0}^{L} 2^{L-i} = 2^{L+1} - 1 = 2N - 1 
\]  

(2.6)

The lone node at the top of the (inverted) tree is the root of the tree. The root is a compact commitment to all nodes.

Every node has a sibling. \( v_6 \) and \( v_7 \) are siblings (with a common parent \( v_{67} \)); likewise, \( v_{86} \) and \( v_{ef} \) are siblings (with a common parent \( v_{8f} \)). Corresponding to any node at level 0 are \( L - 1 \) direct ancestors. For example, the ancestors of node \( v_6 \) are \( v_{67}, v_{47}, v_{07}, \) and \( \alpha \) — one in each level \( 1 \cdots L \). The root \( \alpha \) is a common ancestor for all nodes.

Corresponding to every node in level 0 are \( L \) complementary nodes—one in each level \( 0 \cdots L - 1 \). The \( L = 4 \) complementary nodes of \( v_6 \) are \( v_7, v_{45}, v_{03}, \) and \( v_{8f} \). Note that the complementary nodes of any node includes

1. The sibling of the node
2. The siblings of all ancestors

Together, the nodes complementary to \( v_6 \) can be interpreted as a commitment to all nodes except \( v_6 \). \( v_{8f} \) is a commitment to eight nodes \( v_8 \cdots v_f \); \( v_{03} \) is a commitment to four nodes \( v_0 \cdots v_3 \); \( v_{45} \) is a commitment to \( v_4 \) and \( v_5 \); and \( v_7 \) is a commitment to itself.

Any node in the tree (except the root) is either a right child or a left child of its parent. For example \( v_7 \) is a right child of its parent \( v_{67} \); \( v_{45} \) is a left child of its parent \( v_{47} \). Thus, every node can be associated with an additional bit—say 0 if it is a right-child and 1 if it is a left-child left.

The \( L \) complementary nodes of \( v_6 \) along with their orientations, viz.,

\[
\{(v_7, 0), (v_{45}, 1), (v_{03}, 1), (v_{8f}, 0)\} 
\]
readily provide step by step instructions for mapping leaf $v_6$ to the root, through a sequence of $L$ PRF operations. For example, following the instructions, we can compute the root $\alpha$ starting from $v_6$ as

$$v_{67} = h(v_6 \parallel v_7) \quad v_{47} = h(v_{45} \parallel v_{67})$$
$$v_{07} = h(v_{03} \parallel v_{47}) \quad \alpha = h(v_{07} \parallel v_{8f})$$

Note that the orientation bit specifies the ordering of two nodes before hashing them together to compute the parent node. As $v_7$ is a right-child (orientation 0) it has to be placed to the right of $v_6$ before hashing. Similarly, as $v_{45}$ is a left-child, it has to be placed to the left before hashing.

Also note that the four orientation bits 0, 1, 1, 0 of the complementary nodes of $v_6$ ($v_7$, $v_{45}$, $v_{03}$ and $v_{8f}$ respectively) can be readily obtained from the bits used to represent the index of $v_6$ in binary format (index $6 = 0110_b$). As a second example, the complementary nodes of $v_8$ are

1. Sibling $v_9$ which is a right-child (orientation 0)
2. Sibling $v_89$ of ancestor $v_{ab}$ (orientation 0)
3. Sibling $v_{cf}$ of ancestor $v_{8b}$ (orientation 0)
4. Sibling $v_{07}$ of ancestor $v_{8f}$ (orientation 1)

Once again note that the binary representation of the index $8 = 1000_b$ provides the necessary orientation bits (read from LSB to MSB).

Thus, given any leaf-node $v$ at level 0, it’s index $i$ (where $0 \leq i \leq N - 1$), and the set of its $L$ complementary nodes $c = \{0, \ldots, c_{L-1}\}$, we can define a simple function

$$\alpha = f_{bt}(v, i, c) \quad (2.7)$$

that maps $v$ to the root $\alpha$. The function $f_{bt}()$ can be algorithmically represented as follows:

\[
\begin{align*}
\alpha &= f_{bt}(v, i, \{c_0, c_1, \ldots, c_{L-1}\}) \\
& \quad \text{FOR } (j = 0 \ldots L - 1) \\
& \quad \quad \text{IF } (i \text{ IS EVEN}) \quad v \leftarrow h(v \parallel c_j); \\
& \quad \quad \quad \text{ELSE } \quad v \leftarrow h(c_j \parallel v); \\
& \quad \quad \quad i \leftarrow i >> 1; \quad \text{//right shift by one bit} \\
& \quad \quad \text{RETURN } v;
\end{align*}
\]

As the PRF $h()$ is preimage resistant, it is infeasible to determine alternate values $\tilde{v} \neq v$, and $\tilde{c} \neq c$ that will satisfy $f_{bt}(v, \tilde{c}) = \alpha$.

In applications that employ Merkle trees the root $\alpha$ of the tree is stored in a trusted location. The other $N - 2$ values can be stored in an untrusted location. If values $v$, $c$ received from an untrusted source satisfy $f_{bt}(v, \tilde{c}) = \alpha$, the verifier is convinced of the integrity of such values. More specifically, the verifier is convinced that values $v$ and $c$ were indeed used in the construction of the tree with root $\alpha$. 
2.2 Random Subsets

Several symmetric cryptographic protocols of interest to us in this book are based on the idea of allocation of random subsets of keys [11] from the pool of keys.

Consider a key-pool with $P$ keys $K_1 \cdots K_P$. Let $S_1 \cdots S_N$ represent subsets of $k < P$ keys chosen randomly from the key pool.

Let $k/P = a < 1$. One strategy to choose subset of $k$ keys on an average from a pool of $P$ keys is by picking each key from the pool with probability $a = k/P$. Alternately, if it is desired that each subset should have exactly $k$ keys, the pool of $P$ keys may be divided into $k$ sub-pools, each with $P/k$ keys; from each of the $P/k$ pools one key is picked randomly.

When the key pool and subsets are generated using a PRF $h()$ the generator could start with a single master key $\mu$ to generate the pool keys as

$$K_i = h(\mu \parallel i), q \leq i \leq P.$$  \hspace{1cm} (2.8)

Any subset may be associated with a seed which determines the indexes of the keys chosen to be a part of the subset. For example, for a subset associated with a seed $X$, a random stream of bits generated from repeated application of $h()$ on $X$, for example, $X_1, X_2, \ldots$ generated as

$$X_1 = h(X), X_2 = h(X_2) \cdots$$  \hspace{1cm} (2.9)

can be used to identify the indexes to be assigned to the subset.

Assume that $n$ subsets are picked randomly. Let us represent by $S^n$ the super set of $n$ such subsets. In addition, we randomly choose two other subsets $S_i$ and $S_j$.

Now, two specific questions of interest to us are

1. What is the probability $p$ that all keys contained in a subset $S_i$ is contained in $S^n$?
   a) For a given $n, p$, what is the minimum value of the pool size $P$?
2. What is the probability that all keys in the intersection of $S_i$ and $S_j$ is contained in $S^n$?
   a) For a given $n, p$, what is the minimum value of the pool size $P$?
   b) For a given $n, p$, what is the minimum value of the subset size $k$?

2.2.1 $S_i \subset S^n$

Consider a specific key in the subset $S_i$. The probability that the same key is found in specific subset that was chosen to create $S^n$ is $a$. The probability that the specific key is not found in any of the subsets in $S^n$ is

$$\epsilon = (1 - a)^n$$  \hspace{1cm} (2.10)
Thus, the probability that a specific key in the subset $S_i$ is included in the union of $n$ subsets $(1 - \epsilon)$. Consequently, the probability that all $k$ keys in $S_i$ are included in the union of $n$ subsets is

$$p(n) = (1 - \epsilon)^k = (1 - (1 - a)^n)^k \approx (1 - e^{-an})^P$$  \hspace{1cm} (2.11)$$

Obviously, $p$ increases with $n$. It is often of interest to us to achieve a target $p(n)$ using the least amount of keys. To derive an expression for $P$, Eq. (2.11) can be rewritten as

$$P = \frac{n \log p}{an \log (1 - e^{-an})} = \frac{n \log (1/p)}{-an \log (1 - e^{-an})}$$  \hspace{1cm} (2.12)$$

For a desired $p(n)$ (i.e., if we fix $p$ and $n$), the pool size $P$ is minimized when the denominator $(-an \log (1 - e^{-an}))$ is maximized, which occurs when $an = \log 2$. Corresponding to the choice of $a = \frac{\log 2}{n}$ the maximum value of the denominator is $(\log (1/2))^2 = (\log 2)^2$, and consequently the optimal values of $P$ and $k$ are

$$P = \frac{n \log (1/p)}{(\log 2)^2}$$
$$k = \frac{\log (1/p)}{(\log 2)^2}$$  \hspace{1cm} (2.13)$$

As a numerical example, if we desire $p(n = 1000) = e^{-23} \approx 1 \times 10^{-10}$ (probability of 1 in 10 billion), we choose $a = \frac{\log 2}{1000}$, and

$$P = \frac{1000 \times 23}{\log (2)^2} \approx 47870$$
$$k = Pa = \frac{23}{\log 2} \approx 33.$$  \hspace{1cm} (2.14)$$

In other words, if random subsets each with 33 keys are randomly chosen from a pool of 47870 keys, the probability that the union of 1000 randomly chosen subsets will contain all keys in yet another randomly chosen subset, is about 1 in 10 billion.

### 2.2.2 $(S_i \cap S_j) \subseteq S^n$

Consider a specific key in the pool of $P$ keys. The probability that the key is present in both subsets $S_i$ and $S_j$, and therefore, in $S_i \cap S_j$, is $a^2$. The probability that the key is present in the intersection, but not present in the union $S^n$ is

$$\epsilon = a^2(1 - a)^n.$$  \hspace{1cm} (2.15)$$
Thus, for any of the $P$ keys, the probability that a key is present in the intersection of two sets, and in the union of $n$ sets is $1 - \epsilon$. The probability that all keys present in the intersection are present in the $S^n$ is therefore

$$ p = (1 - \epsilon)^P = (1 - a^2(1 - a)^n)^P \approx (1 - a(1 - a)^n)^k. $$

(2.16)

In other words

$$ P = \frac{\log p}{\log (1 - a^2(1 - a)^n)} $$

(2.17)

$$ k = \frac{\log p}{\log (1 - a(1 - a)^n)} $$

(2.18)

From Eq. (2.18), it can be easily seen that for a given $n$, $p$, the number of keys in each subset, $k$, is minimized when $a(1 - a)^n$ is maximized, which occurs when $a = 1/(n + 1)$. The maximum value of $a(1 - a)^n$ is then

$$ \frac{1}{n + 1} \left(1 - \frac{1}{n + 1}\right)^n = \frac{1}{1 - 1/(n + 1)} \left(1 - \frac{1}{n + 1}\right)^{n+1} \approx \frac{1}{en} $$

(2.19)

Thus, for the optimal choice of $a = 1/(n + 1)$,

$$ p(n) = (1 - \frac{1}{en})^k \approx e^{-k/en} $$

(2.20)

The minimal value $k$ and the corresponding pool size $P = k/a$ are then

$$ k = en \log (1/p) $$

$$ P = en(n + 1) \log (1/p) $$

(2.21)

As a numerical example, if we desire $p(n = 1000) \approx e^{-23}$, we can choose $k = 84974$ and $P = 42487073$.

On the other hand, if we desire to minimize the key pool size $P$, from Eq. (2.18) we can see that it is required to maximize $a^2(1 - a)^n$. This occurs for the choice of $a = 2/n$, corresponding to which the maximum value of $a^2(1 - a)^n$ is

$$ \frac{4}{n^2} \left(1 - \frac{2}{n}\right)^n \approx \frac{4}{n^2} \frac{1}{n^2 e^2} = \frac{4}{n^2 e^2}. $$

(2.22)

As

$$ p(n) = (1 - a^2(1 - a)^n)^P = (1 - \frac{4}{n^2 e^2})^P = e^{\frac{4P}{n^2 e^2}}, $$

(2.23)

we have

$$ P = \frac{n^2 e^2}{4} \log (1/p) $$

$$ k = Pa = \frac{ne^2}{2} \log (1/p) $$

(2.24)

As a numerical example, if we desire $p(n = 1000) \approx e^{-23}$, we can choose $P = 42487073$ and $k = 84974$. 