Merkle Damgard Transform

Sarthak Mahajan, 2018111005

March 7, 2022

1 Merkle Damgard Transform Idea:

The Merkle Damgard Transform constructs a collision resistant hash function H to hash arbitrary length messages given a fixed length collision resistant hash function h which maps messages of size 2n to a hash of size n.Assume we need to hash a message m, where size of m is larger than key size. Let key size be n, IV be an initialization vector. Each message chunk and IV is of size n, if the size of the last chunk is lesser than key length, its padded with zeros to equal key size.Each block is the fixed length collision resistant hash function h. Input to i^{th} block(except the last one is:

$$m_i z_{i-1}$$

, $z_0 = IV$. For the last block, the input is: $|m|z_l$, where l is the number of message chunks and $|m|$ is the message size.The output z_i is a n bit hash.

2 Merkle Damgard Transform Algorithm:

Let (Gen, h) be a fixed-length collision-resistant hash function for inputs of length $2\ell(n)$ and with output length $\ell(n)$. Construct a variable-length hash function (Gen, H) as follows:

- Gen: remains unchanged compared to fixed hash collision resistant function.
- H : on input a key s and a string $x \in \{0,1\}^*$ of length $L < 2^{\ell(n)}$, do the following (set $\ell = \ell(n)$ in what follows):
  - Set $B := \left\lceil \frac{L}{\varepsilon} \right\rceil$ (i.e., the number of blocks in x). Pad x with zeroes so its length is a multiple of ℓ. Parse the padded result as the sequence of ℓ-bit blocks $x_1, \ldots, x_B$. Set $x_{B+1} := L$, where L is encoded using exactly ℓ bits.
  - Set $z_0 := 0^\ell$.
  - For $i = 1, \ldots, B+1$, compute $z_i := h^s\left(z_{i-1} \| x_i\right)$.
  - Output z_{B+1}

3 Merkle Damgard Transform Explanation:

3.1 Theorem:

If (Gen,h) is a fixed length collision resistant hash function, then (Gen, H) is a collision resistant hash function

3.2 Proof:

- Case 1: $L \neq L'$. In this case, the last step of the computation of $H^s(x)$ is $z_{B+1} := h^s\left(z_B \| L\right)$ and the last step of the computation of $H^s\left(x'\right)$ is $z'_{B'+1} := h^s\left(z'_{B'} \| L'\right)$. Since $H^s(x) = H^s\left(x'\right)$ it follows that $h^s\left(z_B \| L\right) = h^s\left(z'_{B'} \| L'\right)$. However, $L \neq L'$ and so $' z_B \| L$ and $z'_{B'} \| L'$ are two different strings that collide for h^s.

- Case 2: $L = L'$. Note this means that $B = B'$ and $x_{B+1} = x'_{B+1}$. Let z_i and z'_i be the intermediate hash values of x and x' during the computation of $H^s(x)$ and $H^s\left(x'\right)$, respectively. Since $x \neq x'$ but $|x| = |x'|$, there must exist at least one index i (with $1 \leq i \leq B$) such that $x_i \neq x'_i$. Let $i^* \leq B+1$ be the highest index for which it holds that $z_{i^*-1} \left\| x_{i^*} \neq z'_{i^*-1} \right\| x'_{i^*}$. If $i^* = B+1$ then $z_B \| x_{B+1}$ and $z'_B \| x'_{B+1}$ are two different strings that collide for h^s because

$$h^s\left(z_B \| x_{B+1}\right) = z_{B+1} = H^s(x) = H^s\left(x'\right) = z'_{B+1} = h^s\left(z'_B \| x'_{B+1}\right).$$

  If $i^* \leq B$, then maximality of i^* implies $z_{i^*} = z'_{i^*}$. Thus, once again, $z_{i^*-1} \| x_{i^*}$ and $z'_{i^*-1} \| x'_{i^*}$ are two different strings that collide for h^s.