Universidad Rey Juan Carlos

Grado en Ingeniería de la Ciberseguridad

Criptografía

Sesión 2. Funciones Hash

---

1. Estudia la resistencia a colisiones de las funciones hash construídas con los siguiente procedimientos: dada una cadena de bits de entrada, $x = b_n b_{n-1} \cdots b_0$

   a) Si z es el entero que representa la cadena x (codificación binaria habitual), se calcula z^2+1, se escribe de nuevo en binario, obteniendo una cadena w se define el hash $H(x)$ tomando los 8 primeros (más significativos) bits de w.

   b) Suponiendo n impar, hacemos $H(x) := b_0 \cdots b_{\frac{n-1}{2}} \oplus b_{\frac{n+1}{2}} \cdots b_n$

   c) Si z es el entero que representa la cadena x, $H(x) = msb_{10}(z \mod 1956)$.

2. Considera la función Hash definida como, dada una cadena de bits x que representa a un número entero z

$$H(x) = msb_4(z \bmod 70) \oplus lsb_2(z \bmod 20)||00.$$

   Por ejemplo, si $x = 10100$, $z = 20$, $H(x) = 1010 \oplus 0000$. ¿Es H resistente a colisiones (CR)? Dada $y = 0100$ ¿Puedes encontrar x con $H(x) = y$?

3. Estudia la resistencia a colisiones (CR) de la función hash construida con el siguiente procedimiento: dada una cadena de bits de entrada, $x = b_n b_{n-1} \cdots b_0$, si z es el entero que representa la cadena x (codificación binaria habitual), se calcula $2z + 2^{25}$, se escribe de nuevo en binario, obteniendo una cadena w y se define el hash $H(x)$ tomando los 8 bits más bajos (menos significativos) de w.

4. Una de las principales aplicaciones de las funciones hash es proporcionar pruebas de integridad. Supongamos que un cliente sube a la nube un archivo X (que borra localmente) y quiere, al bajárselo, verificar que no ha sido modificado. Si guarda un hash de X, podrá luego comparar si el hash del archivo bajado coincide para verificar la integridad del archivo original. Pero, ¿qué pasa si el cliente sube una colección –grande– de archivos? Una solución es calcular un árbol de Merkle, por el que los archivos $X_1, \ldots, X_n$ (con n par) se colocan como hojas de un árbol y vamos construyendo nodos haciendo hashes sucesivos por pares, es decir:

   - hojas (último nivel, t): $h_1 := H(X_1), \ldots, h_n := H(X_n)$
   - nivel $t-1$:

$$h_{12} := H(h_1, h_2)$$
$$h_{34} := H(h_3, h_4)$$
$$\cdots$$
$$h_{(n-1)n} := H(h_{n-1}, h_n)$$

   - nivel $t-2$:

$$h_{1234} := H(h_{12}, h_{34})$$
$$\cdots$$
$$h_{(n-3)(n-2)(n-1)n} := H(h_{(n-3)(n-2)}, h_{(n-1)n})$$

   - ...
   - raíz (nivel 0):

$$h_{12\cdots n}.$$

Responde a las siguientes preguntas:

a) Si $n = 2^t$, ¿cuál es la longitud del mayor camino dentro del árbol de Merkle descrito?

b) Para poder verificar si un archivo X_i no ha sido modificado, es suficiente que el cliente guarde el valor que etiqueta la raíz y que el servidor envie como prueba $\mathcal{O}(t)$ etiquetas del árbol. Razona cómo.