Funciones Hash

Las funciones hash son partes importantes de la criptografía. En esencia son una "huella digital" de longitud fija de un mensaje.

Estas funciones son parte esencial de los esquemas de firmado digital, códigos de autenticación de mensajes, etc.

Motivación

Realmente la que nos va a interesar es la motivación nacida por los esquemas de firmado digital.

Para los esquemas de firmado, algo que realmente "nos saltamos" es el hecho de que estos esquemas tienen un espacio muy limitado para sus mensajes.

¿Cómo entonces se pueden firmar mensajes largos? Una respuesta es por bloques:

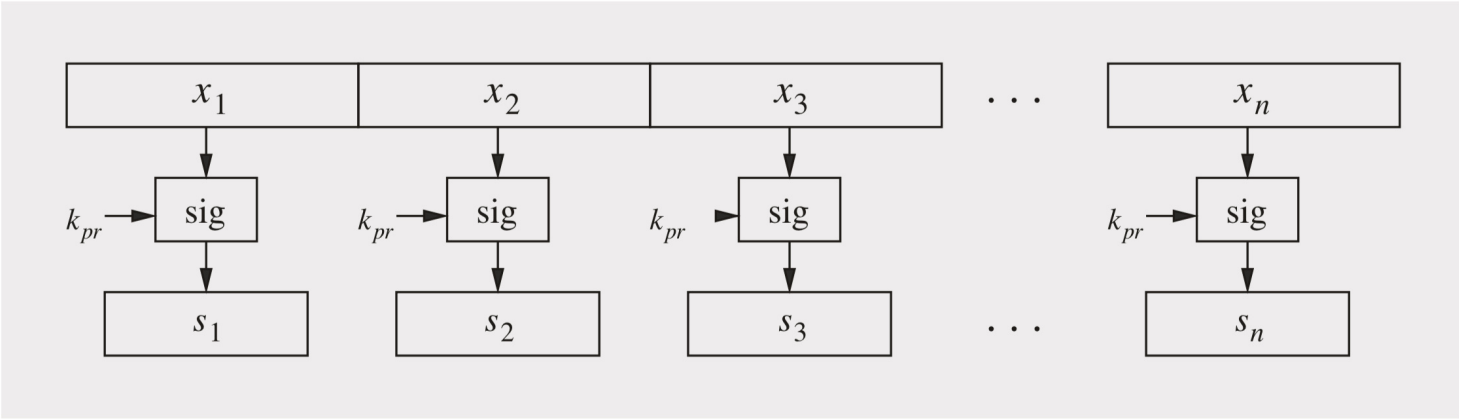


Fig. 11.1 Insecure approach to signing of long messages

pero esto conlleva 3 grandes problemas:

1._ Carga computacional alta

Dependiendo de la longitud del mensaje, la n podría ser muy grande, lo cual hace que se necesiten bastantes recursos computacionales y energéticos tanto para quien firma como para quien revisa la firma.

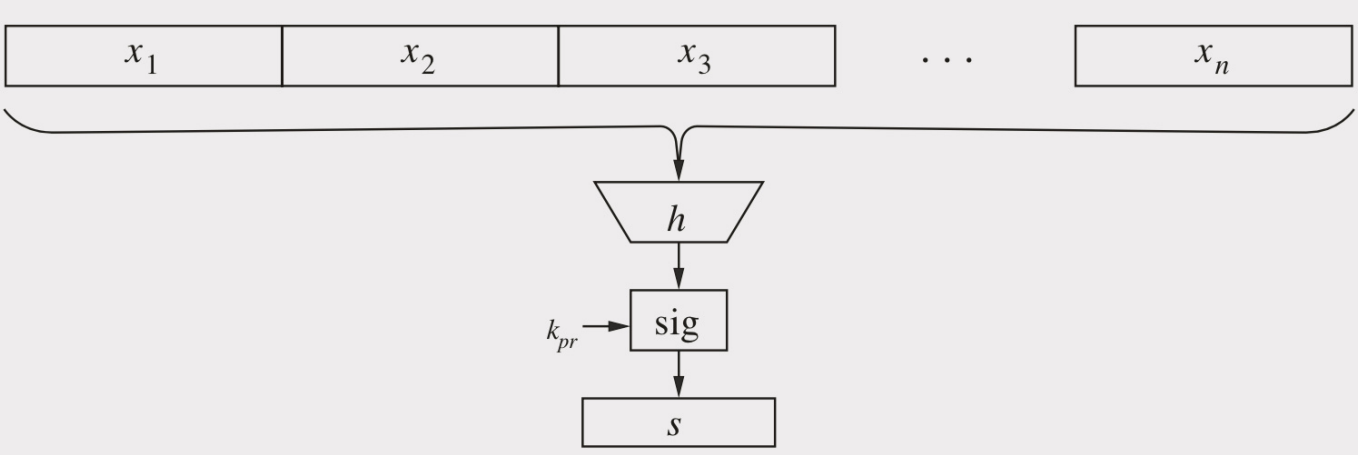
2._ Ancho de banda sobresaturado

No solo se debe mandar el mensaje, sino también las n firmas. Esto puede ser excesivo si n es muy grande.

3._ Limitaciones de seguridad

Mandar los n mensajes permiten nuevas vías de ataque (eliminación de bloques, reordenamiento de bloques, ensamble de nuevos mensajes, etc.)

Con esto en mente, nos gustaría una única firma de corta longitud, para un mensaje de longitud arbitraria. La forma en que se soluciona esto es con las funciones hash:



Con esta función hash se puede tener el siguiente protocolo para un esquema de firma digital:

Aquí x es el mensaje, z su hasheado y s su firma del hasheado.

Basic Protocol for Digital Signatures with Hash Function:

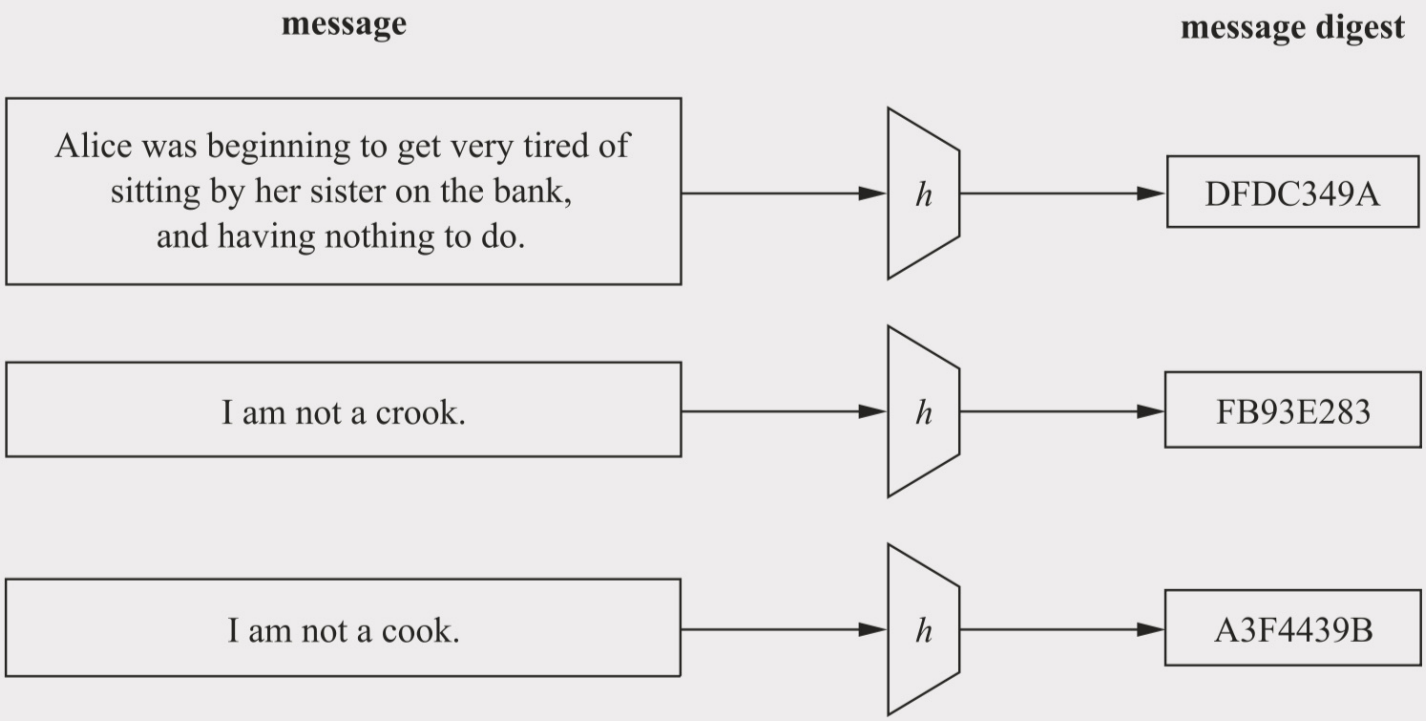
Alice		Bob
	$\xleftarrow{k_{pub,B}}$	
		$z = h(x)$ $s = \text{sig}_{k_{pr,B}}(z)$
	$\xleftarrow{(x,s)}$	
$z' = h(x)$ $\text{ver}_{k_{pub,B}}(s, z') = \text{true/false}$		

Nótese que el firmado y la verificación se realizan usando el hasheado, el cual es referido como message digest o huella digital del mensaje.

En particular, queremos una función matemáticamente eficiente, que admita mensajes arbitrariamente largos y que saque una huella de longitud fija n.

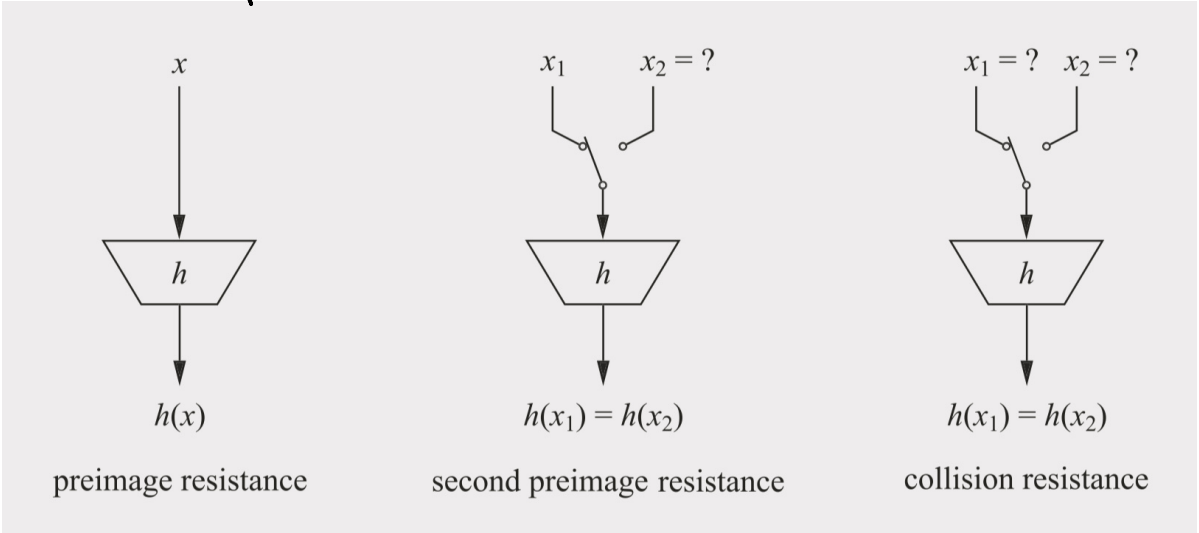
$$h: \{0,1\}^* \longrightarrow \{0,1\}^n \qquad n \sim 256, 512$$

También queremos que sea altamente sensible a cambios en el mensaje.



Requisitos de seguridad

Dado que las funciones hash son bastante generales (claramente no inyectivas) y no tienen llaves, lo que vamos a requerir para su seguridad es lo siguiente.



Resistencia a preimagen

Dada una función hash h y un z en la imagen de h, debe ser computacionalmente imposible encontrar un x tq $h(x) = z$, es decir, cualquier elemento de $h^{-1}(z)$ debe ser computacionalmente imposible de calcular.

La resistencia a preimagen además de lo obvio, también es importante pues podría dársele la vuelta a la seguridad del cifrado a través de la función hash.

Esto es aún más importante en cosas como derivación de llaves.

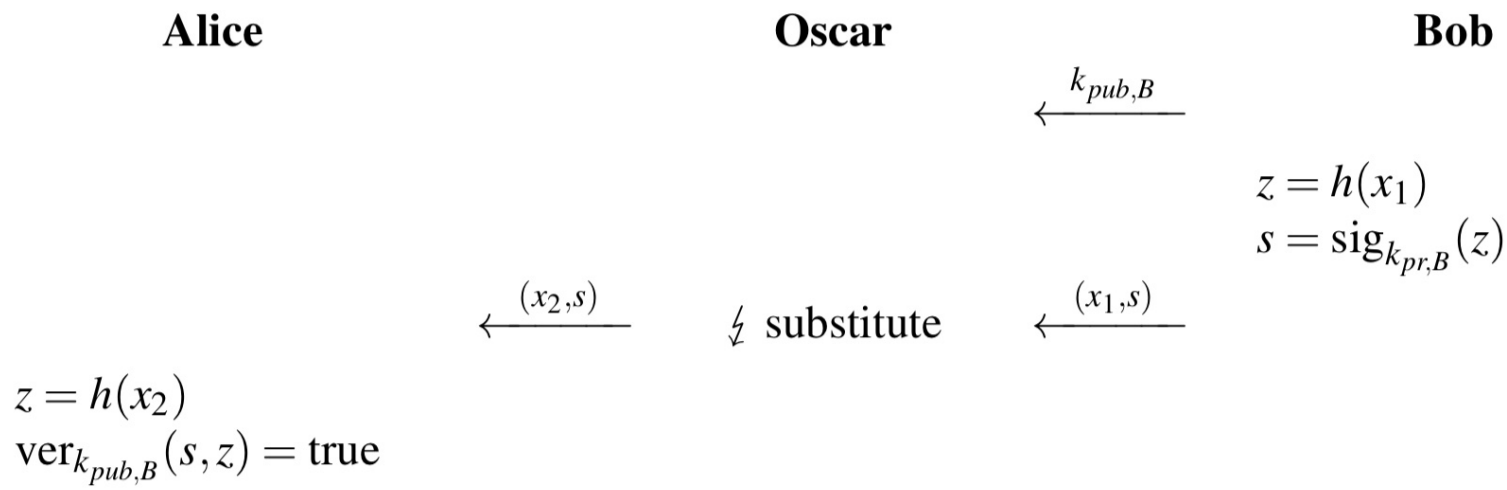
Segunda resistencia a preimagen

En general, para un atacante debe ser imposible obtener dos mensajes $x_1 \neq x_2$ tq $h(x_1) = h(x_2)$ (colisiones).

Si x_1 es dado y x_2 no se conoce se le llama coli-

sión débil.

Incluso en un esquema básico de firmas digitales, es clara la importancia de la resistencia a colisión débil (o segunda resistencia a preimagen).



Por lo tanto, queremos funciones hash h tq dados x_1 y $h(x_1)$ no existen ataques analíticos con los que se pueda construir x_2 con $h(x_2)$.

Para computadoras actuales $n = 128$ basta... por ahora.

Resistencia a colisiones

Una función hash es resistente a colisiones fuertes si no es computacionalmente factible construir $x_1 \neq x_2$

tq $h(x_1) = h(x_2)$.

De no ser así, se podría implementar ataques como el siguiente: Se empieza con x_1' = "Transfiera $10 a la cuenta de Oscar" y x_2' = "Transfiera $10,000 a la cuenta de Oscar", y se alteran los mensajes de forma "no visible" para obtener los dos mensajes con la misma huella digital.

A esto se le llama el ataque del cumpleaños de Yuval (Gideon Yuval, 1979), y depende de que Oscar pueda engañar a Bob para que firme x_1.

Siendo n la longitud de salida de la función hash h, t el número de mensajes, y p la probabilidad de que los mensajes $(x_1, \dots, x_t)$ no produzcan h-colisiones, siguiendo la misma estrategia de la paradoja de los cumpleaños, ent.

$$p \approx e^{-\frac{t(t-1)}{2^{n+1}}}.$$

Luego, haciendo $\lambda = 1-p$ y haciendo varias aproximaciones, tenemos que

$$t \approx 2^{\frac{(n+1)}{2}} \sqrt{\ln \frac{1}{1-\lambda}}$$

$\Rightarrow$ Para encontrar colisiones bastan $t \approx \sqrt{2^n}$ mensajes.

$\Rightarrow$ Para tener seguridad de m bits, es necesario tener $n = (\geq)\ 2m$.

Una tabla para la longitud de n de una función hash.

Table 11.1 Number of required hash computations for a collision for different hash function output lengths and for two different collision likelihoods

	Hash output length [bits]				
λ	128	160	256	384	512
0.5	2^{64}	2^{81}	2^{129}	2^{193}	2^{257}
0.9	2^{65}	2^{82}	2^{130}	2^{194}	2^{258}

Notese que el ataque de cumpleaños es "genérico", es decir, no depende de la función hash específica.

Al final de cuentas, aunque estos tres puntos de seguridad son los más deseados, hay aplicaciones en

las que en realidad basta incluso la resistencia a preimagen como en hasheado de contraseñas.

En resumen:

Properties of Hash Functions

1. **Arbitrary message size** $h(x)$ can be applied to messages x of any size.
2. **Fixed output length** $h(x)$ produces a hash value z of fixed length.
3. **Efficiency** $h(x)$ is relatively easy to compute.
4. **Preimage resistance** For a given output z, it is computationally infeasible to find any input x such that $h(x) = z$, i.e, $h(x)$ is one-way.
5. **Second preimage resistance** Given x_1, and thus $h(x_1)$, it is computationally infeasible to find any $x_2 \neq x_1$ such that $h(x_1) = h(x_2)$.
6. **Collision resistance** It is computationally infeasible to find any pair $x_1 \neq x_2$ such that $h(x_1) = h(x_2)$.