Sécurité Appliquée
Fonctions de hachage

Jean-François COUCHOT
couchot[arobase]femto-st[point]fr

30 janvier 2024

1 Fonction de hachage : exemples et attaques

Exercice 1.1 (Congruence modulo 1024). *On considère la fonction de hachage qui étant donné un nombre retourne son reste dans la division par 1024.*

1. *Formaliser cette fonction $H : B^* \to B^n$.*
2. *Calculer les empreintes de 156 ; 1224 ; 10240.*
3. *Est-elle résistante aux collisions ? Justifier.*
4. *Est-elle résistante à la première préimage ? Justifier.*
5. *Est-elle résistante à la seconde préimage ? Justifier.*

Exercice 1.2 (Résistance aux collisions, mais pas à la première préimage.). *Soit $G : \{0,1\}^* \to \{0,1\}^n$ une fonction de hachage résistante aux collisions. On construit $H : \{0,1\}^* \to \{0,1\}^{n+1}$ définie par*

$$H(m) = \begin{cases} 0||m & \text{si } m \text{ est de longueur } n \\ 1||G(m) & \text{sinon} \end{cases}$$

1. *Montrer que H est résistante aux collisions. On prendra par hypothèse m_1 et m_2 deux chaînes de bits distincts telles que $H(m_1) = H(m_2)$. Raisonner alors sur la taille de m_1 et de m_2.*
2. *Montrer que H n'est pas résistante à la première pré-image.*

Exercice 1.3 (Relation entre la résistance à la seconde préimage et la résistance aux collisions). *Montrer que la résistance aux collisions implique la résistance à la seconde préimage.*

2 Bourrages

Supposons que le message à hacher m a une taille τ_m qui n'est pas un multiple de la longueur du bloc l. Les messages doivent complétés.

Exercice 2.1 (Stratégies naïves de bourrage pour des messages binaires).

1. *On complète le message m par une chaîne de zéros jusqu'à ce que la longueur soit un multiple de l. Montrer que la fonction obtenue (bourrage + hachage) n'est pas résistante aux collisions.*
2. *Supposons maintenant que le processus de bourrage ajoute d'abord un 1 avant de compléter par des zéros comme à la question précédente. Montrer que la fonction obtenue (bourrage + hachage) n'est toujours pas résistante aux collisions.*

Exercice 2.2 (Stratégies de bourrage actuelles pour des messages encodés en octets).

La bibliothèque pycryptodome propose le module `Crypto.Util.Padding` *implantant les méthodes* `pad` *et son homologue* `unpad` *permettant de bourrer (resp. enlever le bourrage) des messages préalablement codés sous la forme d'octets.*

```
def pad(data_to_pad, block_size, style='pkcs7'):
    padding_len = block_size-len(data_to_pad)
    if style == 'pkcs7':
        padding = bchr(padding_len)*padding_len
    elif style == 'x923':
        padding = bchr(0)*(padding_len-1) + bchr(padding_len)
    elif style == 'iso7816':
        padding = bchr(128) + bchr(0)*(padding_len-1)
    else:
        raise ValueError("Unknown padding style")
    return data_to_pad + padding
    else:
        raise ValueError("Unknown padding style")
    return data_to_pad + padding


def unpad(padded_data, block_size, style='pkcs7'):
    pdata_len = len(padded_data)
    if pdata_len == 0:
        raise ValueError("Zero-length input cannot be unpadded")
    if pdata_len % block_size:
        raise ValueError("Input data is not padded")
    if style in ('pkcs7', 'x923'):
        padding_len = bord(padded_data[-1])
        if padding_len<1 or padding_len>min(block_size, pdata_len):
            raise ValueError("Padding is incorrect.")
        if style == 'pkcs7':
            if padded_data[-padding_len:]!=bchr(padding_len)*padding_len:
                raise ValueError("PKCS#7 padding is incorrect.")
        else:
            if padded_data[-padding_len:-1]!=bchr(0)*(padding_len-1):
                raise ValueError("ANSI X.923 padding is incorrect.")
    elif style == 'iso7816':
        padding_len = pdata_len - padded_data.rfind(bchr(128))
        if padding_len<1 or padding_len>min(block_size, pdata_len):
            raise ValueError("Padding is incorrect.")
        if padding_len>1 and padded_data[1-padding_len:]!=bchr(0)*(padding_len-1):
            raise ValueError("ISO 7816-4 padding is incorrect.")
    else:
        raise ValueError("Unknown padding style")
    return padded_data[:-padding_len]
```

1. *Que donnerait le bourrage sur 16 octets du message* `''Secu-L3: le module''.encode('utf-8')` *en prenant la valeur par défaut de style ? On détaillera les lignes exécutées et on expliquera le contenu de chacune d'elles. On notera la liste suivante*

   `[\x00,\x01,\x02,\x03,\x04,\x05,\x06,\x07,\x08,\t,\n,\x0b,\x0c,\r,\x0e,\x0f]`
   qui donne les 16 premieres valeurs retournées par la fonction `bchr`.
2. *Supprimer le bourrage du message* `b'Secu-L3\x08\x08\x08\x08\x08\x08\x08\x08\x08'`.
3. *Les messages* `b'Secu-L3'` *et* `b'Secu-L3\x08'` *vont-ils engendrer le même bourrage ?*
4. *Quel bourrage va-t-il être ajouté si le message a déjà une taille qui est un multiple de* `block_size` *?*