Collision Attack on the Concatenation Combiner

In this exercise, we consider Merkle-Damgård hash functions. For simplicity we do not use any padding (but all of this exercise would apply as well if we used a secure padding scheme).

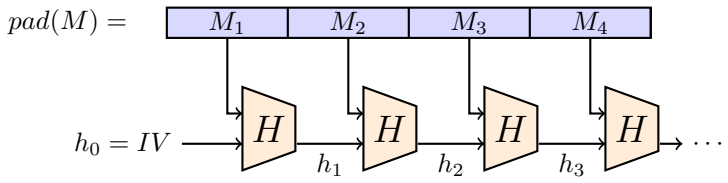


Figure 1: Merkle-Damgård construction.

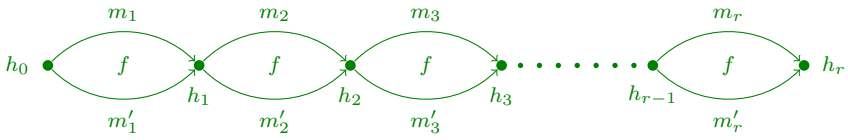
The *concatenation combiner* combines the output of two Merkle-Damgård hashes applied to the same message. These two hash functions F_1, F_2 use different IVs and different compression functions f_1 and f_2. The output of the concatenation combiner is:

$$F(m_0, \ldots, m_{t-1}) = F_1(m_0, \ldots, m_{t-1}) \| F_2(m_0, \ldots, m_{t-1}) \ .$$

We suppose that the chaining values and the outputs of both functions have n bits each. Thus the combiner outputs $2n$ bits. The message blocks will have $2n$ bits, in order to facilitate our attacks.

Question 1. *Show that by using messages of length $n/2$ blocks, one can build a $2^{n/2}$-multicollision of F_1 in time $\mathcal{O}\big(n2^{n/2}\big)$, i.e., a set of messages x_i having all the same length and the same output chaining value. How much space do you need to store this multicollision?*

Solution. *C'est le même principe que la multicollision vue en cours.*



On commence par trouver une collision de f avec une paire de blocs m_1, m'_1. Ensuite en partant de la valeur de sortie, on trouve une collision de f avec une paire de blocs m_2, m'_2. On recommende $n/2$ fois. Il y a $2^{n/2}$ chemins distincts allant de h_0 à $h_{n/2}$, et on a donc $2^{n/2}$ messages distincts ayant tous la même valeur de sortie. De plus ils peuvent bien être tous de la même longueur et on n'a pas de problème de padding.

La multicollision est représentée en mémoire par seulement $n/2$ paires de blocs de message, donc elle ne demande pas beaucoup d'espace. Notons que pour la suite de cet exercice, il sera plus judicieux d'utiliser un peu plus que $n/2 = 16$ blocs successifs (disons 20 ou 24) afin de garantir une très bonne probabilité de succès pour la suite de l'attaque.

Question 2. *Show that we can find a collision of F in time $\mathcal{O}\big(n2^{n/2}\big)$.*

Solution. *On évalue H_2 pour des messages sélectionnés aléatoirement dans la multicollision de la question précédente.*

Comme il y a $2^{n/2}$ messages on s'attend à trouver une collision de H_2 avec probabilité constante. Par définition de la multicollision, H_1 a déjà la même valeur de sortie pour ces deux messages. Donc toute la sortie de H va collisionner entre ces deux messages.

We use the functions `md_hash_1` and `md_hash_2` from the file `tp2_code.py`. Both of them take as input a list of 64-bit integers (the message blocks) and return a 32-bit integer. One uses SHA-2 and the other MD5, so they return completely unrelated outputs. Recall that you can use the function `randrange(1 << 64)` to output a random 64-bit integer.

It is possible to also access the compression function of, say, `md_hash_1`, as follows: `h' = md_hash_1([m], h)` where h is the current chaining value, h' the next one, and m the message block.

Question 3. *Implement the computation of the multicollision of* `md_hash_1`.

Solution. *Pour cette question, le plus simple est de reprendre le code du début du TP1. Supposons que nous en somes au i-ème bloc, et que la valeur de chaînage actuelle est h.*

On crée un dictionnaire d qui contiendra des valeurs de hachage obtenues pour différents messages. On fait une boucle infinie, dans laquelle on sélectionne un bloc au hasard, on calcule `h' = md_hash_1([m], h)` *et on ajoute m dans le dictionnaire à l'index h'. Si h' apparaît déjà dans le dictionnaire, on sort de la boucle, on ajoute la paire de messages trouvés, et on met à jour la valeur de chaînage.*

On répète cette opération autant de fois que l'on souhaite. Je conseille de produire une liste de paires de blocs, de la stocker en dur dans votre code pour les calculs suivants, et de vérifier au passage que des messages obtenus en choisissant des blocs au hasard donnent bien tous la même valeur de `md_hash_1`.

Question 4. *Implement the previous attack on the concatenation combiner* `md_hash_1`$\|$`md_hash_2`*: find a pair of messages m_1, m_2 such that both hash functions have the same output.*

Solution. *On reprend – encore – un code similaire pour la recherche de collisions. Cette fois, les messages aléatoires sont produits en choisissant une série de blocs au hasard dans la liste précédente (pour chaque paire p dans la liste, prendre le bloc* `p[randrange(2)]`*).*

Preimage Attack on the Concatenation Combiner

Question 5. *Show that given a target t, we can find a set of $\mathcal{O}(2^n)$ preimages of t by F_1 in time $\mathcal{O}(2^n)$.*

Solution. *Il faut repartir d'une multicollision à 2^n éléments. En partant de la toute dernière valeur de chaînage, trouver un bloc de message qui la connecte sur la cible t souhaitée. Certes, cette étape demande une recherche exhaustive en temps $\mathcal{O}(2^n)$. Mais par la suite, tout message de la multicollision (en ajoutant le dernier bloc) est une préimage valide.*

Question 6. *Deduce that we can find preimages of the concatenation combiner in time $\mathcal{O}(2^n)$.*

Solution. *Considérons une cible t_1, t_2. On construit d'abord la structure précédente (multicollision + bloc de raccord) qui définit 2^n préimages pour H_1, c'est-à-dire 2^n messages multi-blocs $m_1, \ldots, m_{2^n}$ tels que $H_1(m_i) = t_1$. Ensuite, on fait une recherche exhaustive parmi ces messages pour trouver un m_i tel que $H_2(m_i) = t_2$. Cela nous prendra un temps $\mathcal{O}(2^n)$.*

Concernant la complexité, on peut se contenter ici de la compter en évaluations de la fonction de hachage H_1/H_2. Si on compte en évaluations de la fonction de compression, cela augmenterait le coût à $\mathcal{O}(n2^n)$.

Cependant, on peut parcourir l'ensemble des messages multi-blocs possibles de manière assez intelligente pour avoir une complexité précisément en $\mathcal{O}(2^n)$ (mais je ne vous demanderai pas d'aller jusque dans ces détails).

We now use two different MD functions with 20-bit output, `md_hash_3` and `md_hash_4` (also defined in `tp2_code.py`).

Question 7. *Find a preimage of 0,0.*

Question 8 (Bonus question)**.** *Go back to the two first MD hashes. Append a third MD hash, this time using sha1 and 32 bits of output. Find a collision of the triple concatenation combiner. What is the number of blocks of the colliding messages?*

Solution. *Réutilisez le code de la question pour produire une collision sur les deux premiers hachages MD. Puis une deuxième collision en repartant de la valeur de chaînage, etc. On obtient alors une multicollision simultanée. Les messages sont plus longs : $\mathcal{O}(n^2)$ blocs. On utilise cette multicollision pour trouver une collision sur la troisième fonction.*