

Assignment #2: Solutions

Problem 1

Suppose we can find two message/hash pairs $(M_1, h(M_1))$ and $(M_2, h(M_2))$ such that $M_1 \neq M_2$ and $h(M_1) = h(M_2)$. Then, there exists two distinct Merkle hash trees T_1 and T_2 whose outputs are identical.

We can find a collision for the compression function using a top-down side-by-side comparison of our two tree's, looking across trees for a case where the outputs of f are the same, but the inputs differ. Starting at the root, suppose that either the first block or *msg-length* (thus height) of our trees differ, in this case, clearly we have found a collision in f , so we can stop our search.

We can continue down examining the i th node of T_1 and T_2 and compare the inputs. If they differ, then we have found a collision for the compression function, and the procedure terminates. If not, we proceed to level $i + 1$ and repeat the same procedure. Note that since $M_1 \neq M_2$, we cannot go through the entire tree without finding a collision. It follows that one can find a collision in the compression function.

Problem 2

To construct a collision for the first hash construction with $f_1(x, y) = E(y, x) \oplus y$, choose any y and non-zero x so that $y \neq (x \oplus y)$.

Let:

$$x_1 = D(y, x \oplus y), y_1 = y$$

$$x_2 = D(x \oplus y, y), y_2 = x \oplus y$$

So then:

$$f_1(x_1, y_1) = E(y, x_1) \oplus y = E(y, D(y, x \oplus y)) \oplus y = (x \oplus y) \oplus y = x$$

$$f_1(x_2, y_2) = E(y_2, x_2) \oplus y_2 = E(x \oplus y, D(x \oplus y, y)) \oplus (x \oplus y) = y \oplus (x \oplus y) = x = f_1(x_1, y_1)$$

For the second compression function $f_2(x, y) = E(x, x) \oplus y$, choose any $x_1 \neq x_2$, any y_1 , and let $y_2 = E(x_1, x_1) \oplus E(x_2, x_2) \oplus y_1$.

Then

$$f_2(x_2, y_2) = E(x_2, x_2) \oplus y_2 = E(x_2, x_2) \oplus (E(x_1, x_1) \oplus E(x_2, x_2) \oplus y_1) = E(x_1, x_1) \oplus y_1 = f_2(x_1, y_1)$$

Problem 3

- (a) Since A and all $B_i \in Bs$ share the secret key k , any B_i can send to the parties $B - \{B_i\}$ a message M appended with the MAC under k of M . To the recipients this looks exactly like a message sent by A , and hence, the recipient is not sure the message came from A .

- (b) B_i can successfully fool B_j ($i \neq j$) iff B_i has every key that B_j has. This is easy to see since B_j verifies a message by verifying the MACs corresponding to the keys he has. Thus, for the scheme to work, each pair B_i, B_j ($i \neq j$) must have at least one key not shared between them. Then, no B_i can fool a B_j because B_i will lack one of the keys that B_j uses to verify the message.

Note that we use the assumption of non-collusion to ensure that if some proper subset of parties $\hat{B} \subset B$ have between them every key, they cannot work together to fool the parties in $B - \hat{B}$.

- (c) Let the keys be k_1, k_2, k_3, k_4 . We note that $6 = \binom{4}{2}$. Hence each S_i need only contain two keys. The subsets are:

$$\begin{aligned} S_1: & \{k_1, k_2\} \\ S_2: & \{k_1, k_3\} \\ S_3: & \{k_1, k_4\} \\ S_4: & \{k_2, k_3\} \\ S_5: & \{k_2, k_4\} \\ S_6: & \{k_3, k_4\} \end{aligned}$$

Note that for any two S_i, S_j ($i \neq j$), S_i and S_j differ in at least one element.

Problem 4

- (a) We can construct a new hash from H_1 and H_2 by concatenating them i.e. $H_{12}(M) = H_1(M)||H_2(M)$.

Suppose we have a collision finder for H_{12} i.e. we have a means of finding an M and M' s.t. $M \neq M'$ and $H_{12}(M) = H_{12}(M')$. This clearly also provides collisions in H_1 and H_2 as $H_{12}(M) = H_{12}(M') \Rightarrow H_1(M)||H_2(M) = H_1(M')||H_2(M')$. It follows that $H_1(M) = H_1(M')$ and $H_2(M) = H_2(M')$ by definition of concatenation.

- (b) We can construct a new MAC from M_1 and M_2 by apply each MAC with unique keys (k_1, k_2) and concatenating their outputs i.e. $MAC_{12}(k_1, k_2, M) = MAC_1(k_1, M)||MAC_2(k_2, M)$.

Assume an attack A which is a successful existential forger under a chosen message attack on MAC_{12} , that is, given message tag-pairs $(M_1, t_1), (M_2, t_2), \dots, (M_n, t_n)$ generated by MAC_{12} an attacker can generate a new pair (M', t') that verifies under (k_1, k_2) . The existence of A implies the existence of A_1 and A_2 , that can generate existential forgeries for both MAC_1 and MAC_2 under a similar attack model.

We show the existence of an existential forger A_1 on MAC_1 given A . Given message/tag pairs $(M_1, t_{11}), (M_2, t_{12}), \dots, (M_n, t_{1n})$ for MAC_1 , A_1 generates a random key k_2 and computes $t_{2i} = MAC_2(k_2, M_i)$ for $i = 1 \dots n$, and then passes to A the pairs $(M_1, t_{11}||t_{21}), \dots, (M_n, t_{1n}||t_{2n})$. A then returns the forgery on MAC_{12} , $(M, t_1||t_2)$. A_1 then can return (M, t_1) , which by definition is an existential forgery for MAC_1 . The proof for the existence of A_2 is entirely symmetric. Note that k_1 and k_2 must be independent in the construction MAC_{12} , otherwise insecurity in one of the MACs might leak information about the key allowing an attacker to break MAC_{12} even if one of the MACs is secure.

Problem 5

- (a) For a given $y = g^x$, with x unknown, we know that y is a Quadratic Residue if and only if x is even. Further, by Euler's criteria, we know that y is a QR if and only if $y^{(p-1)/2} = 1 \pmod{p}$. So, if $y^{(p-1)/2} = 1 \pmod{p}$ then we conclude that x is even, and otherwise that x is odd.
- (b) Assuming that x is even, we know that $g^{x/2}$ is a QR if and only if $x/2$ is even. As before, we also know that $g^{x/2}$ is a QR if and only if:

$$(g^{x/2})^{(p-1)/2} = y^{(p-1)/4} = 1 \pmod{p}.$$

This gives us the 2nd least significant bit of x .

- (c) Let $b_n, b_{n-1}, \dots, b_1 \in [0, 1]$ represent the bits of x , and $y = g^x$. Then the following algorithm will recover the bits of x :

```
i = 1
while y ≠ 1
  if y(p-1)/2i = 1 (mod p)
    bi = 0
  else
    bi = 1
  y = y/gbi2i-1
  i = i + 1
```

Note that simply taking the square root and setting $y \leftarrow y^{1/2}$ and using the method of part a) does not work. Indeed if it did, we could compute the discrete log of y for any prime. In fact, the algorithm for taking square roots returns both $\pm y^{1/2}$, which in \mathbb{Z}_p cannot be distinguished as "positive" and "negative" square roots without prior knowledge of the discrete log (Recall $-r = p-r \pmod{p}$). Since only the "positive" root will give the correct next bit, the method fails.

- (d) This algorithm does not work for a random prime p because at each step i , we are relying on $(p-1)/2^i$ being an integral value. Although for a random odd prime, the argument in part (a) still holds.