TD 4 - Design of a provably secure hash function

I. Design of a hash function $\{0,1\}^{2m} \longrightarrow \{0,1\}^m$

1. *$p-1 = 2q$ and q is prime; so, the divisors of $p-1$ are $\{1, 2, q, 2q = p-1\}$.*
Since d is a divisor of $p-1$, we have $d \in \{1, 2, q, p-1\}$.

2. *Since $0 \leq x_2, x_4 \leq q-1$: $-(q-1) \leq x_4 - x_2 \leq q-1$.*
But q is prime; then $(x_4 - x_2)$ is prime to q and lesser than q, so $d \neq q$; and, since $p-1 = 2q$, $d \neq p-1$.

3. *Obvious:* $\alpha^{x_1}\beta^{x_2} \equiv \alpha^{x_3}\beta^{x_4} \mod p \Longleftrightarrow \alpha^{(x_1-x_3)} \equiv \beta^{(x_4-x_2)} \mod p$

4. *If $d = 1$, let $u = (x_4 - x_2)^{-1} \mod (p-1)$: $u.(x_4 - x_2) = 1 + k.(p-1)$ Then $\beta^{(x_4-x_2).u} \mod p \equiv \beta^{1+k(p-1)} \mod p \equiv \beta \mod p$ (from Fermat's little theorem).*
Replacing in 3., we obtain : $\beta = \alpha^{(x_1-x_3).u} \mod p$, i.e. $\lambda = (x_1 - x_3).u \mod p-1$, qed.

5. **5.a.** *Since $d = 2$ and $p-1 = 2.q$, we have $x_4 - x_2$ prime to q; so $u.(x_4 - x_2) = 1 + k.q$.*
Then $\beta^{(x_4-x_2).u} \mod p \equiv \beta^{1+kq} \mod p \equiv \beta.(\beta^q)^k \mod p$.
But $q = \frac{p-1}{2}$ and β is a primitive elements mod p. Thus, $\beta^{p-1} = 1 \mod p$ and $\beta^q = \beta^{\frac{p-1}{2}} = -1 \mod p$. Finally, $\beta^{(x_4-x_2).u} = (-1)^k.\beta \mod p$, qed.
5.b. *Replacing in 3., we have : $\beta = \pm\alpha^{(x_1-x_3).u} \mod p$ ie $\beta = \alpha^{(x_1-x_3).u+\delta.q} \mod p$ with $\delta \in \{0,1\}$. Thus, either $\delta = 0$, i.e. $\lambda = u.(x_1 - x_3) \mod p-1$ or $\delta = 1$, i.e. $\lambda = u.(x_1 - x_3) + q \mod p-1$, qed.*

6. *From previous questions, we have the following algorithm:*

```
AlgoCalculLogBeta( p, α, β, ;x1, x2, x3, x4 ) {
  q = (p − 1)/2;
  d = pgcd(x4 − x2, p − 1) ;
  if (d == 1) {
       u = (x4 − x2)^(−1) mod (p − 1);
       λ = (x1 − x3).u mod p − 1;
  }
  else {// here d == 2
       u = (x4 − x2)^(−1) mod q;
       λ = (x1 − x3).u mod p − 1;
       if (ExpoMod( α, λ, p ) == −β) λ = λ + q ;
  }
  return λ ;
}
```

The cost is $O(1)$ arithmetic operations mod $p-1$, p and q; thus $O(\log^{1+\epsilon} p)$, which is small even for large values of p (eg 1024 bits). So, if a collision is known for h_1, Then we may easily compute the discrete logarithm β, which is in contradiction with the hypothesis that λ is very expensive to compute. Thus h_1 is collision resistant.

II. Extension to a hash function: $\{0,1\}^* \longrightarrow \{0,1\}^m$

7.

$$\begin{array}{rlcl} h_2 : & (\{0,1\}^m)^4 & \to & \{0,1\}^m \\ & (x_1,x_2,x_3,x_4) & \mapsto & h_1(h_1(x_1,x_2),h_1(x_3,x_4)) \end{array}$$

8. *Let $x \neq y$ be a collision for $h_2 : h_2(x) = h_2(y)$. We distinguish two cases:*

- *either $h_1(x_1,x_2) \neq h_1(y_1,y_2)$ or $h_1(x_3,x_4) \neq h_1(y_3,y_4)$: thus, since $h_1(x_1,x_2),h_1(x_3,x_4)) = h_1(y_1,y_2),h_1(y_3,y_4))$ we found a collision on h_1.*
- *or, since $x \neq y$, we may by symmetry restrict to the case $(x_1,x_2) \neq (y_1,y_2)$. Then, since $h_1(x_1,x_2) = h_1(y_1,y_2)$, we have a collision on h_1.*

All computations are performed in $O(m)$ time –comparisons here-, which is polynomial (linear here) in the input (x,y) size.
Since h_1 is assumed collision resistant, we deduce by contradiction that h_2 is collision resistant too.

9. *By induction, we state that if h_i is collision resistant, then h_{i+1} is collision resistant too.*

- *Base case: for $i = 1$, h_1 is assumed collision resistant.*
- *Induction: similarly to previous question, we prove that if h_{i+1} is not collision resistant, then h_i is not collision resistant; the proof is exactly the same, just replacing h_1 by h_i and h_2 by h_{i+1}.*

Since h_1 is collision resistant by hypothesis, then h_i is collision resistant for any $i \geq 2$.

10. *Let $C(i)$ be the number of calls to h_1 performed during computation of h_i. We have $C(i) = 2.C(i-1)+1 = 2^i.C(0) + \sum_{k=0}^{i-1} 2^k = 2^i - 1$.*
For a n bits sequence, we thus call n/m times h_1. The cost of h_1 is $\tilde{\Theta}(m)^{1+\epsilon})$. Then the cost is then $O(n.m^\epsilon) = O\left(n^{1+\epsilon}\right) = \tilde{O}(n)$.

11. *Let A ne the message and n its number of bits. To compute $H(A)$, let i such that $2^i.m = n$ i.e. $i = \lceil log_2 \frac{n}{m} \rceil$. Then we compute $H(A) = h_i(A)$.*
Using recursion, this algorithm may also be used on-line to hash an input bit stream (i.e. the size n of the message is discovered when EOF is met).
Another alternative is to use the Merkle-Damgard protocol (cf lecture).

III. HAIFA Extension scheme

12

13

14⋆ M2R assignment