# Homework 5

Erik Brakke

October 15, 2015

**Collaborators:** Alison Kendler, Kyle Hogan, Isaac Cohen.

## Answer 1

Proof: We know that $f$ is a permutation
Thus, we know that the last $2k$ bits of $f'$ are a permutation on $\{0,1\}^{2k}$
The first $k$ bits are all independent of each other, making the total possibilities of the first $k$ bits $2^k$
This means that $f'$ has $2^{3k}$ possible outputs
$x$ always has an output for $f$, and $c$ can always be found for $x$
This means that any $x$ of length $3k$ always has an output for $f'$
For $x \neq y$, either $f(x) \neq f(y)$ or $c(s_x) \neq c(s_y)$
It is possible for the minority bits to all be in the same positions, but if $x \neq y$, then the majority bit must be different
It is possible for the majority bits to all be the same, but if $x \neq y$, then the minority bit must be in a different position
This means that $f'(x)$ has a unique output in $\{0,1\}^{3k}$
This means that $f'$ is a permutation

Now I will prove the one-wayness of $f'$ with a reduction
Let's assume that $f'$ was not a one-way function. This means that there is an algorithm $D$ such that given $f'(x)$, $D$ could compute $x$ in polynomial-time
We can use the fact that we can reverse $f'$ to reverse $f$
Given $f'(s)$ we can use $D$ to find $s$.
Once we have $s$, we can compute easily compute $d(s_1) \circ ... \circ d(s_k)$
These are the inputs to $f$, meaning that using $D$ we can reverse $f$
However we assumed that $f$ was a one way functions, therefore $f'$ must be a one way function as well

$\square$

Given a bit position $i$, one can compute $i//3$ to figure out $c$ by looking at the $i//3$ position of $f'(x)$. This is the majority bit of the group of 3 that $i$ is in
If the majority bit is 1, then $\Pr[i = 1] = 3/4$ because out of the 4 possible combinations where $c = 1$, 1 appears 9 times out of the 12 bits
The same logic holds for $c = 0$
Because computing $i//3$ runs in polynomial time, and worst case you would have to search $k$ positions, the algorithm runs in linear time

## Answer 2

We can use CRT to compute $c_{DB} = m^2 mod(n_d * n_b)$
We know that if $m < \sqrt{n}$, then given $c$, m is just the integer square root of $c$
$m$ must be less than $\sqrt{n_d * n_b}$ because $m < min(n_d, n_b)$
$min(n_d, n_b) < \sqrt{(}n_d * n_b)$
So now we can just take the integer square root of $c_{DB}$ to get $m$

## Answer 3

Let's assume we have an algorithm $R$ that can predict $SK$ with non negligible probability
Now we can use $R$ to build a distinguisher $D$ that can break the definition of secure multi-bit encryption
$D$ chooses two messages $m_0, m_1$ after seeing $PK$
$D$ then uses $R$ to compute $SK$ from $PK$ with non negligible probability
When $D$ is presented with $Enc_{PK}(m)$, he can run $Dec_{SK}(Enc_{PK}(m))$ to obtain $m$
He can then distinguish between $m_0, m_1$ with non-negligible probability
This contradicts our assumption that (Gen,Enc,Dec) is a secure public key crypto system
Therefore, such an $R$ cannot exist

$\square$

---

## References

None