1. Hash Functions

Intuition: You have a large object that you want to represent with just a few bits.

Suppose you want to hash a map of Providence. There’s a limit to how much you can compress it without losing information.

We want a function $H$, such that given $m = H(\text{map of PVD})$, $\text{Sig}_{\text{CoP}}(m)$ can’t find another $\text{map}'$ s.t. $H(\text{map}') = m$.
(where $\text{Sig}_{\text{CoP}}$ means “signature from the city of Providence.”)

Formalization:
A family of collision resistant hash functions (crhf) consists of efficient algorithms $\text{Gen}$, $H(\cdot, \cdot)$ s.t. $\text{Gen}(1^k)$ outputs a key $PK$.

$H_{PK} : \{0, 1\}^* \rightarrow \{0, 1\}^k$ s.t. $\forall$ ppt $A$, $\exists$ negligible $\nu(k)$ s.t.

$$\Pr[(x_1, x_2) \leftarrow A(1^k, PK) : x_1 \neq x_2 \text{ and } H_{PK}(x_1) = H_{PK}(x_2)] \leq \nu(k)$$

A universal one way hash function (family) (UOWHF, pronounced “woof” ¯

$\dashv$) consists of algorithm(s) $H : \{0, 1\}^* \rightarrow \{0, 1\}^k$ (and $\text{Gen}$ if it’s a family) st. $\forall$ ppt $A$, and $\forall$ positive polynomials $p(\cdot)$

$$\Pr[x_1 \leftarrow \{0, 1\}^{p(k)}; x_2 \leftarrow A(x_1) : x_1 \neq x_2 \text{ and } H(x_1) = H(x_2)] \leq \nu(k)$$

If it is a family, then $\text{Gen}$ generates a PK inside the experiment, the adversary gets to see it, and $H$ is indexed by the PK.

A uowhf can be constructed from a one way function.

Why could something be a uowhf but not a collision resistant hash function? The idea is that in the collision resistant hash function experiment, you get to choose your inputs. so maybe there is only a collision on a negligible fraction of the inputs but if you know something about what those inputs look like, you can find it. Whereas, in the uowhf experiment, the input is chosen at random, so the chance that it’s a “bad” input is small. In fact you can’t construct one from the other.

However, notice that a uowhf isn’t good enough for our toy “map of providence” example, because someone could design the map so that it was vulnerable to collisions. (the mob could totally do it.)

Here is what NIST uses:
what they used to use: SHA (this has been broken)
Right now, SHA-1 is still acceptable. (160 bits) but people believe it will be broken. So
NIST is currently holding a contest for designing a new hash function. Before it was broken MD5 was very popular. (stands for “message digest 5,” was invented by Ron Rivest)

2. Constructions

Input length 2\(k\) \(PK = \) group \(G\) of prime order \(p \approx 2^k\) (i.e. its bit length is \(k\) and it starts with a 1.), \(g, h \in G\)
\(H_{PK}(x_1, x_2) = g^{x_1} h^{x_2}\) where \(x_1, x_2 \in \mathbb{Z}_p\)

Suppose \(A\) finds a collision on input \(PK\). Then we can use \(A\) to solve the discrete log problem.

\(B\) gets \(G, p, g, h\) and forwards it to \(A\).
If \(A\) finds a collision: \(x_1, x_2, y_1, y_2\) s.t. \(g^{x_1} h^{x_2} = g^{y_1} h^{y_2}\)
we have \(g^{x_1-y_1} = h^{y_2-x_2}\).
suppose \(y_2 - x_2 \neq 0 \mod p\)
then \(g^{x_1-y_1} = h\), so \(B\) has found an \(\alpha\) s.t. \(g^\alpha = h\) and we’ve broken discrete log.
however, if \(y_2 - x_2 = 0 \mod p\), then \(x_1 - y_1 = 0 \mod p\) and \(A\) has not actually found a collision.

Input length that’s a multiple of a power of 2
Suppose we have: \(h_{PK}\{0,1\}^{2^k} \rightarrow \{0,1\}^k\) and want \(H_{PK}\{0,1\}^* \rightarrow \{0,1\}^k\) A natural idea is to iterate. Say you have \(x\) of length \(k \cdot 2^\ell\), i.e. \(x = x_0 x_1 \ldots x_{2^\ell}\)
\(x^j = h_{PK}(x^0 \ldots x^{j-1})\) This construction is called a Merkle Tree. You will see it in the homework. You will prove \(h_{PK}^{\ell} : \{0,1\}^{2^k} \rightarrow \{0,1\}^k\) is collision resistant. (but we still don’t have something that is collision resistant for arbitrary length input.)

Arbitrary input length
Revised construction (Merkle-Damgard) \(H_{PK}(x\) of polynomial length \(p\).

\(1\) let \(\ell = \lceil \log(\frac{p(k)}{k}) \rceil\).
Pad \(x\) with \(2^\ell k - p(k)\) 0’s to obtain \(x'\)
\(y = h_{PK}^{\ell}(x')\)
\(2\) write \(p\) in binary using \(k\) bits. output \(h_{PK}(p \circ y)\).
This one works.

3. Commitment Scheme

Remember our motivating example for hard core bits:
To commit to bit \(b\), find \(x\) s.t. \(B(x) = b\) and publish \(y = f(x)\). where \(f\) is a one way permutation. The committer could be unbounded but still not be able to change its mind. However, the scheme only works if the receiver is poly time.

Using a crhf, you can have a scheme where the committer must be poly time, but the
receiver can be unbounded. Let $h$ be a crf \( \{0,1\}^{2k} \to \{0,1\}^k \) Let $B : \{0,1\}^{2k} \to \{0,1\}$ be a “balanced” boolean function. (outputs just about as many 0’s as 1’s)

To make a commitment, find an $x$ s.t. $B(x) = b$. Output $H(x)$. Now, the committer cannot change his mind unless he can find a collision. Even if the receiver is unbounded, though, since $B$ is balanced, he cannot make any guesses about what $x$ was.