Chapter 2.1 : The Power of Randomness: Reed-Solomon Fingerprinting

We can combine Reed-Solomon codes with fingerprinting to generate robust corruption and error resistant codes.

Reed-Solomon Codes

These are type of error-correcting codes that work by transforming a data sequence such that the data can be recovered even if some part of data is corrupted.

Fingerprinting

In context of data, fingerprint is a unique identifier representing some data. Fingerprinting can generate unique or nearly unique fingerprints for some distinct data.

Randomness

Randomization of data can help with generation of distinct fingerprints to reduce the overall predictability of some cryptographic application.

Example

Setting

Imagine two users, Alice and Bob, holding some very large files consisting of n ASCII characters (thus m = 128 possible characters). We can denote the content of Alice's file as a sequence of characters (a1, a2…an) and Bob's file as (b1, b2…bn). The goal here is to determine if their files are equal while minimizing communication for any reason (large file, security, etc.).

Our goal is to allow Alice and Bob to execute a randomized procedure that may output a wrong value with a tiny probability (say 0.0001). Then they can solve this problem with a really small amount of communication.

Communication Protocol

n = # of ascii characters in the files

m = 128 possible ascii characters

$a1,\ a2....an$ = content in Alice's file

$b1,\ b2....bn$ = content in Bob's file

High level idea is that Alice is going to pick some hash function h at a random from some family of hash functions H. Here, $h(a)$ would be a very short fingerprint of a. This means that for any $a\ !\ =\ b$, fingerprints for both differ with a high Probability over a random choice of h. Now Alice can send h and $h(a)$ to Bob and Bob can check if $h(a)\ =\ h(b)$.

Details

p = some prime number $p\ \ >=\ \ max\{m,\ n^2\}$ <-- choose a prime number p that is greater than or equal to maximum of 2 numbers: m (which is 128) or n^2 (square of ASCII characters in dataset)

F_p = set of integers modulo p (for all ops performed on numbers in this set, if result is p or greater, we divide it by p and take the reminder). Refer to this link to learn the modular arithmetic.

Here, the reason why p must be choosen larger than n^2 is because of the error probobility of the protocol is less than n/p and we want this quantity to be bounded by $1/2$ (larger p = smaller error probability).

The protocol requires selecting a prime ParseError: KaTeX parse error: $ within math mode, due to the way it interprets inputs from Alice and Bob. These inputs are conceptualized as vectors in the finite field F_p^n. Every character from Alice's and Bob's files must have a unique corresponding value in F_p. To cover all possible characters (since m represents the total number of ASCII characters), it's important to choose p larger than m. This way, the finite field can properly and uniquely represent every character.

Notation

So, For each $r \in F_p$, $h_r(a_1...a_n)$ = $\sum_{i=1}^{n} a_i.r^{i-1}$ **Eq. 1**

The family H of considered hash functions is

$H = \{h_r : r \in F_p\}$ **Eq. 2**

Explanation and simplification of Notation

EQ2 : We have a different hash function h_r for each value r in some field F_p. This hash function interprets its input param $(a_i...a_n)$ as coefficients of a degree $n - 1$ and outputs the polynomial evaluated at r.

EQ 1 : So, we can get the hash by taking sum after multiplying each input a_i with r^{i-1}.

So Alice picks a random element r from F_p, computes $v = h_r(a)$ and sends v and r to Bob. Bob outputs EQUAL if $v = h_r(b)$ and NOT-EQUAL otherwise.

Analysis

This protocol can now output the correct answer with a very high probability. Particularly:

- If $a_i = b_i$ for all $i = 1.....n$, then Bob outputs equal for every choice r.
- If there is even one i such that $a_i! = b_i$, then Bob outputs NOT-EQUAL with probability atleast $1 - (n - 1)/p$, which is atleast $1 - 1/n$ by choice of $p \geq n^2$.

Let $p_a(x) = \sum_{i=1}^{n} a_i.x^{i-1}$ and similarly $p_b(x) = \sum_{i=1}^{n} b_i.x^{i-1}$. We can see that both p_a and p_b are polynomials in x of degree at most $n - 1$. If there is even one i such that $a_i \,! = b_i$, then there are at most $n - 1$ values of r such that $p_a(r) = p_b(r)$. Since r is choosen at random, the probability that Alice picks such r is $(n - 1)/p$. Thus Bob outputs NOT-EQUAL with probability atleast $1 - (n - 1)/p$ (where probability is over random choice of r).

Cost of Protocol

Alice sends only 2 elements of F_p to Bob, namely v and r. This indicates that number of transmitted bits grows logarithmically with respect to n which is $O(log\ n)$ assuming that $p \leq n^c$ for some constant c.

Feel free to visit this link to learn the **order of growths**.

To study more about **finite fields**, refer to this link.

Final Notes

We refer to above protocol as Reed-Solomon fingerprinting because $p_a(r)$ is a random entry in an error-corrected encording of the vector $(a_i....a_n)$. The encoding is called Reed-Solomon encoding.