1 Week 1

1.1 Multiway paper

This week, I mainly worked on the Ahlswede's [?] paper. Here is a summary of what I did:

- The "$\sqrt{n}$ trick" is described in [?]. The idea behind the section F's 3 step algorithm is also described in previous works of Ahlswede. I will take a look at the referenced article next week.
- I think the section B. can be described more intuitively as follows:
  - Ω is the set of *terminals* which are basically the communication devices.
  - Γ is the set of *messengers* which can be viewd as a pair of transmitter and receiver. For each messenger γ, $\mathcal{N}_\gamma$ is the set of messages that the transmitter and receiver communicate with.
  - For each terminal $\omega \in \Omega$, the set of transmitters on that terminal is denoted by $\mathcal{A}_\omega$.
  - For each terminal $\omega \in \Omega$, the set of receiver on that terminal is denoted by $\mathcal{B}_\omega$. — there is a typo in the definition given in the paper, the last $\mathcal{B}_\omega$ should be changed to ω.
  - For each terminal $\omega \in \Omega$, the set of available feedback lines is denoted by Φ_ω. — there is a typo here too, $\Phi_\omega \subset \Omega$ and not Γ.
  - The channel W is discrete and memoryless.

  the given assumption can also be interpreted as follows

  - $\mathcal{A}_\omega \cap \mathcal{B}_\omega = \emptyset$: because otherwise, the transmitter and receiver would be placed on the same terminal which makes communication via channel unnecessary.
    $\cup_{\omega\in\Omega}\mathcal{A}_\omega = \cup_{\omega\in\Omega}\mathcal{B}_\omega = \Gamma$: we can assume that each transmitter/receiver is placed only on one terminal.
  - If $|\mathcal{X}_\omega| = |\mathcal{Y}_\omega| = 1$, then terminal can not transmit or receive any information.
  - If $|\mathcal{X}_\omega|$ then the terminal can not send information, hence no transmitter should be placed on it. Similarly for receiving.
  - I did not fully understand what is logic behind A_4 but I guess that is related to relay channels, since the relays do not send or decode data.
  - $\omega \in \Phi_\omega$ every terminal should know what it received.
  - If $\gamma \in \mathcal{A}_\omega \cap \mathcal{B}_{\omega'}$, then the transmitter of γ is on ω and its receiver is on ω'. Then, all the information available at ω' is feedbacked to ω, i.e. $\Phi_{\omega'} \subset \Phi_\omega$.
  - Passive decoders do not need to transmit anything.
- On section C:
  - Randomized feedback is not explained, I dont see what makes them different than the stochastic feedback defined later.
  - "concatentation of strategies" after equation (1.6) is ambiguous.
  - Derivation of equation (1.8) might be something like the following, but I am not sure as the definition are not formal.

$$\begin{aligned}
\mu(\mathcal{F}_{m+n}) &= \max_{f^{n+m}\in\mathcal{F}_{n+m}} H(Y^{n+m}(f^{n+m})) \\
&\geq \max_{f^{n+m}\in\mathcal{F}_n\times\mathcal{F}_m} H(Y^{n+m}(f^{n+m})) \\
&= \max_{f^n\in\mathcal{F}_n}\max_{f^m\in\mathcal{F}_m} H(Y^n(f^n), Y^m(f^m)) \\
&= \max_{f^n\in\mathcal{F}_n}\max_{f^m\in\mathcal{F}_m} H(Y^n(f^n)) + H(Y^m(f^m)) \\
&= \mu(\mathcal{F}_n) + \mu(\mathcal{F}_m)
\end{aligned}$$

    I used independence in the second and the third line.
  - Given the above inequality, $\mu(\mathcal{F}_n) \geq n\mu(\mathcal{F}_1)$ and therefore

$$\mu((\mathcal{F}_n)_{n=1}^{\infty}) = \lim_{n\to\infty}\frac{1}{n\mu(\mathcal{F}_n)} \leq \lim_{n\to\infty}\frac{1}{n^2\mu(\mathcal{F}_1)} =?0$$

    and since $\mu \geq 0$, then $\mu = 0$??!.
  - I skimmed the remaining sections. I am not sure how the mystery number μ is related to the 3-step algorithm.

1.2 Randomized prime

I also worked on the randomized prime generation idea. Consider the following algorithm Instead of checking that

Algorithm 1: Generating random primes

```
input  : $n, t$
output: A uniform $n$-bit prime
for $i = 1 \to t$ do
    $p \leftarrow \{0, 1\}^n$
    if $p$ is prime then
        return $p$
    end
end
return $\perp$
```

p is a prime, we can check if p passes the Miller-Rabin test or not, which is more efficient – running in $\mathcal{O}(n^3)$ rather than in $\mathcal{O}(2^n)$ for a simple primality test, or $\tilde{\mathcal{O}}(n^6)$ for AKS primality test. By applying the Miller-Rabin test multiple times, the probability of error (a composite number passes the test) decreases rapidly. Moreover, by letting n to be large enough (an asymptotic formula can be derived from the Prime Number Theorem), we can be sure that π_K can be represented by n-bits, and therefore our analysis for the 3-step algorithms remains unchanged.

For the next week I am going to do a more thorough derivation of the above idea and implement it.

2 Week 2

Add the randomized prime generation and a testing python code to plot the error rate.

3 Week 3

3.1 The simulation code

3.1.1 `Channel` class

```
typedef function<uint64_t(uint64_t)> ChannelFunc;
typedef uint64_t chnl_output;
typedef uint64_t chnl_input;
class Channel
{
private:
    uint64_t x, y; /* size of the input and output alphabet
     X = {0, 2, ... , x-1} , Y = {0, 2, ... , y-1} */
    ChannelFunc* f; // the randomized function of channel

public:
    Channel(uint64_t x, uint64_t y, ChannelFunc* f);
    ~Channel();
    chnl_output transmit(chnl_input symb); // returns the result of
       tranmission of symb
};
```

The `channel` class models a discrete memoryless channel. The `ChannelFunc` is a function that takes an input character – the characters are modeled as indicies– and returns an output character. This function might be randomized as well. In fact the channel's transition matrix W should be implemented in `ChannelFunc` and be given as input to the constructor. It was wiser to let indicies start from 0 and I will refactor the code.

The `transmit` method simply calls the `ChaanelFunc f` on the given symbol.

3.1.2 `Identification` class

```
using namespace std;

typedef function<vector<chnl_input>* (mpz_t)> ID_EncodingFunction; // encoder
typedef function<long double (const vector<chnl_output> &)>
   ID_DecodingFunction; // decoder
typedef function<bool (const vector<chnl_output> &, mpz_t)>
   ID_IdentifiyingFunction; // identifier
// typedef tuple<ID_EncodingFunction* ,ID_DecodingFunction*,
   ID_IdentifiyingFunction*> ID_Code;



class IdentificationCode
{
private:
    uint64_t loglog_number_of_messages; // log of log of number of messages
    uint64_t block_length; // block length
    uint64_t number_of_encoding_iteration; // number of encoding iteration


    ID_EncodingFunction* encoder;
    ID_DecodingFunction* decoder;
    ID_IdentifiyingFunction* identifier;
    long double first_error ;
    long double second_error;
    bool valid_construction;

public:
    IdentificationCode(uint64_t loglog_number_of_messages, uint64_t
       block_length, uint64_t number_of_encoding_iteration);
    ~IdentificationCode();
    void constructID_Code(const Channel & channel, function<void (const
       Channel& , IdentificationCode* )>* construction_method);
    uint64_t getLogLogNumberOfMessages();
    uint64_t getBlockLength();
    uint64_t getNumberOfEncodingIteration();
    void setBlockLength(uint64_t block_length);
    long double getFirstKindError();
    long double getSecondKindError();
    void setSecondKindError(long double second_error);
    void setEncoder( ID_EncodingFunction * enc);
    void setDecoder( ID_DecodingFunction * dec);
    void setIdentifier( ID_IdentifiyingFunction * idn);
    vector<chnl_input>* encode(mpz_t message); // encodes the message
    long double decode(const vector<chnl_output> &received); // gives the log
       of the number of messages the could be identified with {received}
    bool identify(const vector<chnl_output> &received, mpz_t message); // does
        the {received} identifies {message}
};

typedef function<void (const Channel& , IdentificationCode* )>
   ID_CodeGenerator;
```

This class models an identification code for a given channel `C`. The construction of the codes is done by `constructID_Code` which takes a `construction_method` to do the job. The `encoder` is a function that takes a message and encodes to a block of channel's inpute character. The `decoder` returns **the number of identified messages**. Under a uniform distribution on messages, the second error rate – false identification – is equal to the number of identified messages divided by the number of messages N. I have not implemented the `getFirstKindError` and `getSecondKindError` yet.

There is a similar `transmission` class which is supposed to model transmission codes. I added this class because of the constructions in [?] and [?] that use transmission codes. However, I have only implemented the 3-step algorithm so far.

3.1.3 `Codes` class

```
void NoiselessBSC_ID(long double alpha,const Channel &channel,
    IdentificationCode * id_code); // the 3 step coding scheme
```

For now it only implements the 3-step algorithm.

3.1.4 `Simulate` class

```
pair<uint64_t, long double>  simulate(Channel & channel,uint64_t
    loglog_number_of_messages , uint64_t block_length, uint64_t
    number_of_encoding_iterations,
       ID_CodeGenerator* construction_method);
/* randomly generate a messages from N= {0,2, ... , N-1} and  transmit it over
     the given channel with the given code*/
```

It takes as input a `Channel C` and a identification code constructor `construction_method` then, it simulates the transmission of a message over channel. For now, it outputs the second error rate of the Identification code.

3.1.5 main

```
using namespace std;
using namespace std::chrono;

uint64_t round(uint64_t dividend, uint64_t divisor){
    uint64_t integer_part = dividend / divisor;
    uint64_t fraction_part = dividend % divisor;
    if(fraction_part < dividend >> 1){
        return integer_part;
    }
    return integer_part + 1;
}

int main(int argv, char *argc[])
{
    uint64_t loglog_number_of_messages = (argv >= 2) ? stoull(argc[1]) : 4;
    uint64_t block_length = (argv >= 3) ? atoi(argc[2]) : 0;
    uint64_t number_of_simulation = (argv >= 4) ? atoi(argc[3]) : 30;
    uint64_t number_of_encoding_iteration = (argv >= 5) ? atoi(argc[4]) : 2;
    long double alpha = (argv >= 6) ? stold(argc[5]) : 1.1;
    file_address = (argv >= 7) ? argc[6] : "D:/Backup/Codes/
       IdentificationChannel/logs/log-default.txt";
    random_seed = (argv >= 8) ? argc[7] : "";
    gmp_random_seed = (argv >= 9) ? argc[8]: "";

    init_gmp_generator();
    mpq_t avg_error;
    mpq_init(avg_error);
    mpq_canonicalize(avg_error);
    uint64_t avg_block_length = 0;
    high_resolution_clock::time_point start,end;
    uint64_t avg_time;

    for (uint64_t i = 0; i < number_of_simulation; i++)
    {
        start = high_resolution_clock::now();
        uint64_t result =  simulate_nonprime(loglog_number_of_messages,
           number_of_encoding_iteration,alpha,avg_error);
        end = high_resolution_clock::now();

        avg_time += duration_cast<std::chrono::microseconds>(end-start).count
           ();
        avg_block_length += result;
    }
    mpq_t q_num_sims;
    mpq_init(q_num_sims);
    mpq_set_ui(q_num_sims,number_of_simulation,1);
    mpq_div(avg_error,avg_error, q_num_sims);
    cout <<  round(avg_block_length,number_of_simulation) << "␣" << mpq_get_d(
       avg_error) << "␣" << avg_time / number_of_simulation;
    *getOutputStream() << "LogLog␣of␣Number␣of␣Messages:" <<
       loglog_number_of_messages << endl;
    *getOutputStream() << "End\n" << endl;
    getOutputStream()->flush();
    getOutputStream()->close();
    return 0;
}

/*
    Channel noiselessBSC = Channel(2, 2, new ChannelFunc([](chnl_input x)
                                                         { return x; }));
    init_gmp_generator();
    double avg_error = 0;
    uint64_t avg_block_length = 0;
    high_resolution_clock::time_point start,end;
    uint64_t avg_time;
    ID_CodeGenerator* NoiselessCode = generateNoiselessBSC_ID(alpha);

    for (uint64_t i = 0; i < number_of_simulation; i++)
    {
        start = high_resolution_clock::now();
        pair<uint64_t,double> result =  simulate(noiselessBSC,
           loglog_number_of_messages, block_length,
           number_of_encoding_iteration, NoiselessCode);
        end = high_resolution_clock::now();
        avg_time += duration_cast<std::chrono::microseconds>(end-start).count
           ();

        avg_block_length += result.first;
        avg_error += result.second;
    }
*/
```

The `main` method takes some optional inputs

1. Address of a file for logging.
2. The parameter N, the number messages.
3. The parameter n, the block size – for 3-step algorithm is not important.
4. The parameter m, the number of simulations.
5. A random seed for initializing the random generator.

3.2 Random Prime Generation

The pseudocode of our prime generation algorithm is

Algorithm 2: pseudocode

```
input  : positive integers $m, s, k$
output: A uniformly chosen odd prime less than or equal to $m$
for $i = 1 \to s$ do
    $n \leftarrow \{3, 5, \dots, m\}$
    if $Miller\_Rabin(n, k)$ is PRIME then
        return $n$
    end
end
return 23
```

Instead of returning $\perp$ when all of the randomly chosen numbers are COMPOSITE, it returns 23. Therefore, the probability of finding no prime number is

$$\begin{aligned}
\mathbb{P}\{\perp\} &= \mathbb{P}\{M\_R(n_1, k) = \text{COMPOSITE}, \dots, M\_R(n_s, k) = \text{COMPOSITE}\} \\
&= \prod_{i=1}^{s} \mathbb{P}\{M\_R(n_i, k) = \text{COMPOSITE}\} \\
&= \prod_{i=1}^{s} \mathbb{P}\{n_i \text{ is composite}\} \\
&= \prod_{i=1}^{s} (1 - \mathbb{P}\{n_i \text{ is prime}\}) \\
&\approx \prod_{i=1}^{s} \left(1 - \frac{1}{\ln m}\right) \\
&= \left(1 - \frac{1}{\ln m}\right)^{s}
\end{aligned}$$

where we used the fact that $\frac{\pi(m)}{m} \approx \frac{1}{\ln m}$ asymptotically by prime number theorem [?]. From Theorem 31.39 of [?] and its following analysis, for moderate values of $s \approx 3$, the probability of error is negligible. That is, if the algorithm returns a number – not $\perp$– then it is most likely a prime. Then, the number of iteration to get a prime number is about

$$\begin{aligned}
&\mathbb{P}\{\perp\} \leq \frac{1}{2} \\
\Longrightarrow\ & s \lg \left(1 - \frac{1}{\ln m}\right) \leq -1 \\
\Longrightarrow\ & s \geq \frac{-1}{\lg\left(1 - \frac{1}{\ln m}\right)} \\
\Longrightarrow\ & s \geq \frac{-\ln 2}{\ln\left(1 - \frac{1}{\ln m}\right)}
\end{aligned}$$

note that $\ln(1 - \frac{1}{x}) \approx -\frac{1}{x}$ for large enough x

$$\Longrightarrow s \geq \ln m \ln 2 = \ln^2 2 \lg m$$

which means that s is in the order of number of bits of m.

4 Week 4

Identification is a communication paradigm introduced by Ahlswede [?]. In identification schemes, in essence, the receiver wants to know whether a certain message has been send or not. This is in contrast to the Shannon's transmission paradigm where the receiver wants to know the content of the message.

More formally, the send and receiver both have the message set $\mathcal{M}$ and the receiver is interested in message $m \in \mathcal{M}$. Ofcourse, when the sender knows m, he can send a bit to indicate that he intends to send m or not. We may then assume that sender does not know m.

This problem can be trivially addressed by transmission codes, the receiver decodes the received code to $\hat{m}$ and then decides if $\hat{m} = m$. However, the Ahlswede's identification codes require exponentially shorter blocklength to identify the same number of messages. This improvement is achieved mainly by relaxing the condition that the decoding sets need be disjoint. By allowing the decoding sets to have slight overlap, Ahlswede [?] has shown that there exists coding schemes that can identify $2^{2^{nC}}$ messages where C is the Shannon capacity of the DMC channel.

There are two kinds of errors associated with an identification scheme. The first kind happens when the sender sends m but the receiver fails to identify it and hence *misses* the identification. The second kind happens when the sender send $m' \neq m$ and the receiver *falsely* identifies m instead.

Definition (Identification code): An identification code $(n, N, \lambda_1, \lambda_2)$ for a DMC channel $\mathcal{W}^{\setminus}(\mathcal{X}^n|\mathcal{Y}^n)$ is a set $\{Q(\cdot|i), \mathcal{D}_i\}_{i\in[N]}$ where $Q(\cdot|i)$ is a distribution over $\mathcal{X}^n$ to that encodes i – for determinstic encoder $Q(x_i|i) = 1$ for some $x_i \in \mathcal{X}^n$, and $\mathcal{D}_i \subset \mathcal{Y}$ is the decoding set of i. The first and second kind errors are bounded by λ_1 and λ_2, respectively.

$$P_{e,1}(i) = \sum_{x^n\in\mathcal{X}^n} Q(x^n|i)W^n(\mathcal{D}_i^c|x^n) \leq \lambda_1$$

$$P_{e,2}(i,j) = \sum_{x^n\in\mathcal{X}^n} Q(x^n|j)W^n(\mathcal{D}_i|x^n) \leq \lambda_2$$

For all $1 \leq i, j \leq N$ and $j \neq i$.

4.1 Prime Number Generator

Prime number generators are algorithms that generate primes (I guess :))) well technically the name is self-explanatory). There are multitude of PNG, each might be desirable given the requirements of the problem. Typically we may consider two types of PNG; prime sieves and primality tests. In a prime sieve we look for all primes in a given interval $[m, n]$ where usually $m = 1$. One of the simplest –but not the most efficient– sieves is the sieve of Eratosthenes. Prime sieves are not very efficient, even if there was no computation, just outputing the primes less than n is of order $O(n/\ln n)$, which makes these algorithms sub-exponential in size of n.

Algorithm 3: sieve of Eratosthenes

```
input  : positive integer $n$
output: All prime numbers less than or equal to $n$
create a bitset $b$ of size $n$ set initially to all true
$b[1]$ = false
$P = \emptyset$
for $i = 2, \ldots, n$ do
    if $b[i]$ is true then
        $P = P \cup \{i\}$
        for $j = 2i, \ldots, \lfloor \frac{n}{i} \rfloor i$ do
            $b[j]$ = false
        end
    end
end
return $P$
```

However, in most application we look for some large primes and not all primes. One strategy to generate such primes is to pick a random number – perhaps from a desirable distribution– and then check whether that number is a prime. Therefore, we need an efficient primality test algorithm. There are determinstic and randomized polynomial time primality tests. However, currently the best determinstic algorithms are much slower than randomized ones – higher polynomial degree– and at the same time, the randomized algorithms can be made very accurate. As a result, for most applications a randomized test is used. A list of basic primality test algorithms can be found in [?].

4.2 3-Step Algorithm

In essence the 3-Step algorithm does the following:

1. The sender uniformly chooses two indicies k, l from $[K], [K']$ respectively.
2. Given any message $m \in [M]$, the sender send $\phi_l(\phi_k(m))$ to the receiver.

I think if we send π_k and π_l instead of the indicies can signficantly improve the efficiency of the receiver without compromising the rate. Note that, $\pi_k = O(k \ln k)$ – the constant term is relatively small $C \leq 30$ for all integers and $C \leq 15$ for $k \geq 200$. Therefore, the asymptotic transmission cost of this modified scheme is

$$\begin{aligned} n &= \lceil \lg \pi_K \rceil + 2 \lceil \lg \pi_{K'} \rceil & (1) \\ &= O(\lg K + \lg\lg K) + O(\lg K' + \lg\lg K') & (2) \\ &= O(\lg K) + O(\lg K') \sim \alpha[1 + o(1)] \lg\lg M & (3) \end{aligned}$$

which is the same as before. The benefit of this modification is that the receiver does not need to compute π_k and π_l from k and l respectively – as far as I know only a prime sieve can do this reliably, however as explained above, the sieves are subexponential.

I also came up with a new error analysis which I think is better than the Ahlswede's. First consider the following lemmas

Lemma 1. *For any $x \geq 2$,*

$$\frac{1}{x} \sum_{p \leq x} \frac{1}{p} = \frac{\lg\lg x}{x} + O(\frac{1}{x}) \tag{4}$$

The approximate term can be expanded in terms of Mertens' constant.

Lemma 2. *Let $m, m' \leftarrow [M]$ and $k \leftarrow [K]$ be a uniform random variable independent of m, m'. Then,*

$$\mathbb{P}\left(\phi_k(m) = \phi_k(m') | m \neq m'\right) \leq \frac{\lg\lg K}{K} + O(\frac{1}{K})$$

Proof. We easily have

$$\begin{aligned} \mathbb{P}\left(\phi_k(m) = \phi_k(m') | m \neq m'\right) &= \sum_{\hat{k}=1}^{K} \mathbb{P}\left(\phi_k(m) = \phi_k(m') \middle| m \neq m', k = \hat{k}\right) \mathbb{P}\left(k = \hat{k} \middle| m \neq m'\right) \\ &= \frac{1}{K} \sum_{\hat{k}=1}^{K} \mathbb{P}\left(\phi_k(m) = \phi_k(m') \middle| m \neq m', k = \hat{k}\right) \\ &\leq \frac{1}{K} \sum_{\hat{k}=1}^{K} \frac{1}{\pi_{\hat{k}}} \\ &= \frac{\lg\lg K}{K} + O(\frac{1}{K}) \end{aligned}$$

Now by following the error analysis of Ahlswede we get

$$\mathbb{P}\left(\phi_l(\phi_k(m)) = \phi_l(\phi_k(m')) | m \neq m'\right) \leq \frac{\lg\lg K}{K} + \frac{\lg\lg K'}{K'}$$

and we have

$$\frac{\lg\lg K}{K} = \frac{\lg \alpha + \lg\lg M}{(\lg M)^{\alpha}} \leq \frac{1}{(\lg M)^{\alpha - 1}}$$

I came up with this when I was working on the error analysis of our code, where we used the Miller-Rabin to generate prime. I am currently working on the distribution of the generate numbers and I will add it as soon as I become confident.

5 Week 5

5.1 Miller-Rabin analysis

Miller-Rabin is a well-known random primality test algorithm. Let $MR(n, k)$ be the distribution of Miller-Rabin algorithm on the prime candidate n with k repeats. Let $\mathcal{P}$ be the set of primes. Then, we have the following probabilities.

$$hi \tag{5}$$

$$MR(n,k) = \begin{cases} 1 & \text{with probability 1, if } n \text{ is prime} \\ 1 & \text{with probability less than} 4^{-k}\text{, if } n \text{ is composite} \\ 0 & \text{with probability more than} 1-4^{-k}\text{, if } n \text{ is composite} \end{cases}$$

Let $GMR(n,s,k)$ be prime number generator that uses Miller-Rabin test. It can be shown that,

$$\mathbb{P}(GMR(n,s,k) = \perp) \simeq (1-\frac{1}{m})^s(1-4^{-k})^s$$

If we bound this error probability with ϵ, then we get the following bound on s.

$$s \geq -m \ln \epsilon$$

The probability that the result of $GMR(n,s,k)$ is composite, given it is not $\perp$ is as follows.

$$\mathbb{P}(GMR(n,s,k) \text{ is composite} | GMR(n,s,k) \neq \perp) \leq s(1-\frac{1}{m})4^{-k}$$

If we bound this error probability with δ, then we get the following bound on s.

$$s \leq 2\delta 4^k$$

Let $\epsilon = e^{-l}$ and $\delta = 2^{-q}$ with $l, q \geq 0$. Then,

$$ml \leq s \leq 2^{2k+1-q}$$

Note that, $s = ml$ and $k = \dfrac{\lg(ml)+q}{2}$ satisfies both inequalities.

5.2 Prime Number Theorem

Theorem 3 ([?]). *Let $\pi(x)$ denote the number of primes less than or equal to $x \geq 1$. The prime number theorem states that*

$$\lim_{x\to\infty} \frac{\pi(x)\ln x}{x} = 1$$

That is, $\pi(x) \sim \dfrac{x}{\ln x}$. This implies that n-th prime p_n is asymptotically given by $p_n \sim n \ln n$.

Moreover, we may frequently use these exact bounds on $\pi(x)$.

Lemma 4 ([?]). *For any $n \geq 2$,*

$$\frac{1}{6}\frac{n}{\ln n} \leq \pi(n) \leq 6\frac{n}{\ln n}$$

and for any $n \geq 1$,

$$\frac{1}{6} n \ln n \leq p_n \leq 12(n \ln n + n \ln \frac{12}{e})$$

5.3 Simulation Code

I also worked on our code. There was a mistake where I outputed the curve of number of message M to second kind error λ_2. Given that $n \sim \lg \lg M$, at the time I thought this fine. But now the code produces to curves; n vs M and n vs λ_2.

Because of the double exponential nature of M, the code will fail on larger values of M. However, we can we use the fact that the algorithm is mostly concerened about $\lg M$ and alleviate this problem.

6 Week 6

6.1 3-step algorithm

The 3-step algorithm as described in [?] defines the following parameters.

- Let $\mathcal{M} = \{1, 2, \ldots, M\}$ be the message set and $\alpha > 1$. Let $K = \lceil (\lg M)^\alpha \rceil$.
- Let π_i denote the i_{th} prime. Let $\mathcal{M}' = \{1, 2, 3, \ldots, \pi_K\}$ and $K' = \lceil (\lg \pi_K)^\alpha \rceil$.

graphic/MessageVsBlocklength.png

Figure 1: The number of message vs block length curve

graphic/ErrorVsBlocklength.png

Figure 2: The second kind error vs block length curve

- Let us denote the set $\{1, 2, \ldots, \pi_l\}$ by $\mathbb{Z}_l^+$. Define the function $\phi_l : \mathbb{N} \to \mathbb{Z}_l^+$ as follows.
$$\phi_l(n) = [n \mod \pi_l] + 1 \tag{6}$$
Where $[n \mod \pi_l]$ is equal to the remainder of the division of n by π_l.

A round of communication in this scheme is as follows.

1. The sender chooses a key $k \leftarrow \mathcal{K} = \{1, 2, \ldots, K\}$ uniformly and transmits it.
2. The sender chooses another key $l \leftarrow \mathcal{K}' = \{1, 2, \ldots, K'\}$ uniformly and transmits it.
3. Given a message $m \in \mathcal{M}$, the sender transmits $\phi_l(\phi_k(m))$. Assuming that receiver wishes to identify $\hat{m} \in \mathcal{M}$, he calculate $\phi_l(\phi_k(\hat{m}))$ and compares it with $\phi_l(\phi_k(m))$. He identifies the message as $\hat{m}$ whenever $\phi_l(\phi_k(m))$ $= \phi_l(\phi_k(\hat{m}))$.

Example 1. Suppose $M = 100$ and $\alpha = 1.5$. Then, $K = 18$, $\pi_K = 61$, and $K' = 15$. Assume that $k = 12$ and $l = 7$ are chosen, with $\pi_k = 31$ and $\pi_l = 17$. Given $m = 71$, the sender sends the following code.
$$\underbrace{1100}_{k=12}\underbrace{111}_{l=7}\underbrace{01011}_{\phi_l(\phi_k(71))=11}$$
We immediately find this scheme to be problematic when the sender sends the code all at once. There needs to be some separator that allows the receiver to determine where each part of the code starts and ends. However, let us continue with the example. If $\hat{m}_1 = 71$, then clearly the receiver correctly identifies the message. If $\hat{m}_2 = 32$
$$\phi_l(\phi_k(\hat{m}_2)) = 16 = (10000)$$
and the receiver correctly does not identify the message. However, when $\hat{m}_3 = 9$, $\phi_l(\phi_k(\hat{m}_3)) = 11 = (01011)$ which means the receiver falsely identifies $m = 71$ as $\hat{m} = 9$.

Theorem 5 ([?]). *The 3-step algorithm produces a $(n = n(M, \alpha), M, \lambda_1 = 0, \lambda_2 = \lambda_2(M, \alpha))$ identification code – per* **??***– such that $\lambda_2 \to 0$ as $M \to \infty$. Moreover, this coding scheme is optimal.*
$$\lim_{M \to \infty} \frac{\lg \lg M}{n(M)} = \frac{1}{\alpha} \tag{7}$$
Proof. It is obvious that $\lambda_1 = 0$, that is the receiver will not miss an identification. For the second kind error probability consider the following lemmas.

Lemma 6. *Any positive integers m has at most $\lfloor \lg m \rfloor$ unique prime factors.*

Proof. Suppose $p_1, \ldots, p_k$ are all the prime factors of m. Then for some $\alpha_1, \ldots, \alpha_k \geq 1$
$$m = \prod_{i=1}^{k} p_i^{\alpha_i} \geq \prod_{i=1}^{k} 2^{\alpha_i} \geq 2^k$$
As a result, $k \leq \lfloor \lg m \rfloor$ as required. ■

Lemma 7. *For any $m, m' \in \mathcal{M} = \{1, 2, \ldots, M\}$ such that $m \neq \hat{m}$*
$$\left| \left\{ k \in \{1, 2, \ldots, K\} \,\middle|\, \phi_k(m) = \phi_k(\hat{m}) \right\} \right| \leq \lg M \tag{8}$$
Proof. The given set is the set of common prime factors of m and $\hat{m}$ that are less than or equal to π_K. The inequality immediatly follows from the fact that $m, \hat{m} \leq M$ and M has at most $\lg M$ prime factors. ■

We can derive an upper bound for the second kind error.
$$\begin{aligned}
P_{e,2}(m, \hat{m}) &= \Pr[\phi_l(\phi_k(m)) = \phi_l(\phi_k(\hat{m})) \mid m \neq \hat{m}] && (9)\\
&= \Pr[\phi_l(\phi_k(m)) = \phi_l(\phi_k(\hat{m})) \mid \phi_k(m) = \phi_k(\hat{m}), m \neq \hat{m}] \Pr[\phi_k(m) = \phi_k(\hat{m}) \mid m \neq \hat{m}] && (10)\\
&\quad \Pr[\phi_l(\phi_k(m)) = \phi_l(\phi_k(\hat{m})) \mid \phi_k(m) \neq \phi_k(\hat{m}), m \neq \hat{m}] \Pr[\phi_k(m) \neq \phi_k(\hat{m}) \mid m \neq \hat{m}] && (11)\\
&\leq \Pr[\phi_k(m) = \phi_k(\hat{m}) \mid m \neq \hat{m}] + \Pr[\phi_l(\phi_k(m)) = \phi_l(\phi_k(\hat{m})) \mid \phi_k(m) \neq \phi_k(\hat{m})] && (12)\\
&\leq \frac{\lg M}{K} + \frac{\lg M'}{K'} && (13)\\
&= \frac{\lg M}{\lceil (\lg M)^{\alpha} \rceil} + \frac{\lg \pi_K}{\lceil (\lg \pi_K)^{\alpha} \rceil} && (14)\\
&= \frac{1}{(\lg M)^{\alpha - 1}} + \frac{1}{(\lg \pi_K)^{\alpha - 1}} && (15)
\end{aligned}$$

By the prime number theorem, **??**, $\pi_K \sim K \ln K$. As a result, $\lambda_2 \to 0$ as $M \to \infty$.

$$(\lg \pi_K)^{\alpha-1} \sim (\lg K + \lg\lg K - \lg\lg e)^{\alpha-1} \approx (\alpha \lg\lg M + \lg\lg\lg M + \lg \alpha - \lg\lg e)^{\alpha-1}$$

Finally, the blocklength is calculated by $n = \lceil \lg K \rceil + \lceil \lg K' \rceil + \lceil \lg \pi_{K'} \rceil$. By applying the prime number theorem **??**

$$\begin{aligned}
n &= \lceil \lg K \rceil + \lceil \lg K' \rceil + \lceil \lg \pi_{K'} \rceil \\
&= \lceil \lg \lceil (\lg M)^{\alpha} \rceil \rceil + \lceil \lg \lceil (\lg \pi_K)^{\alpha} \rceil \rceil + \lceil \lg \pi_{K'} \rceil \\
&\approx \alpha \lg\lg M + \alpha \lg\lg \pi_K + \lg \pi_{K'} \\
&\approx \alpha \lg\lg M + (1 + o(1)) \lg\lg\lg M + (\alpha + o(1)) \lg\lg \pi_K \\
&\approx \alpha (1 + o(1)) \lg\lg M
\end{aligned}$$

which was what was wanted. ■

This scheme requires sender to have access to a prime generation algorithm that given K computes the first K primes. And it requires the receiver to have access to another prime generation algorithm that given an index k output π_k, the k_{th} prime. To the best of our knowledge both these algorithm are exponential (I think the best algorithm are subexponential in fact)[needs referencing]. To alleviate these inefficiencies we propose the following modifications.

6.2 Modified 3-step algorithm

Consider the following parameters.

- Let $\mathcal{M} = \{1, 2, \dots, M\}$ be the message set and let $K = \lceil \lg M \rceil$, $K' = \lceil \lg K \rceil$, and $C = (\lg M)^{\alpha}$ for some constant $\alpha > 0$.
- Let us denote the set $\{1, 2, \dots, l\}$ by $\mathbb{Z}_l^+$. Define the function $\phi_l : \mathbb{N} \to \mathbb{Z}_l^+$ as follows.

  $$\phi_l(n) = [n \mod l] + 1 \tag{16}$$

  Where $[n \mod l]$ is equal to the remainder of the division of n by l.

A round of communication in this scheme is as follows.

1. The sender chooses a probabilistic prime k from the set $\mathcal{K} = \{1, 2, \dots, \lceil CK \lg K \rceil\}$ by a prime number generator and transmits it.
2. The sender chooses another probabilistic prime l from the set $\mathcal{K}' = \{1, 2, \dots, \lceil CK' \lg K' \rceil\}$ by the same prime number generator and transmits it.
3. Given a message $m \in \mathcal{M}$, the sender transmits $\phi_l(\phi_k(m))$. Assuming that receiver wishes to identify $\hat{m} \in \mathcal{M}$, he calculates $\phi_l(\phi_k(\hat{m}))$ and compares it with $\phi_l(\phi_k(m))$. He identifies the message as $\hat{m}$ whenever $\phi_l(\phi_k(m))$ $= \phi_l(\phi_k(\hat{m}))$.

Example 2. Suppose $M = 100$ and $C = 6$, then $K = 7$, $CK \lg K = 118$, $K' = 3$, and $CK' \lg K' = 29$. Assume that $k = 67$ and $l = 13$. Given $m = 71$, the sender sends the following code.

$$\underbrace{1000011}_{k=67}\underbrace{1101}_{l=13}\quad\underbrace{0110}_{\phi_l(\phi_k(71))=6}$$

We again require that the codes be separated by time or by some special character. If $\hat{m}_1 = 71$, then clearly the receiver correctly identifies the message. If $\hat{m}_2 = 32$

$$\phi_l(\phi_k(32)) = 8 = (1000)$$

and the receiver correctly does not identify the message. However, when $\hat{m}_3 = 4$, $\phi_l(\phi_k(4)) = 6 = (0110)$ which means the receiver falsely identifies $m = 71$ as $\hat{m} = 6$.

Theorem 8. *This coding scheme is optimal.*

$$\lim_{M \to \infty} \frac{\lg\lg M}{n(M)} = \frac{1}{1 + 3\alpha} \tag{17}$$

Proof. The blocklength is calculated by $n = \lceil \lg|\mathcal{K}| \rceil + 2\lceil \lg|\mathcal{K}'| \rceil$. By applying the prime number theorem **??**

$$\begin{aligned}
n &= \lceil \lg|\mathcal{K}| \rceil + 2\lceil \lg|\mathcal{K}'| \rceil \tag{18}\\
&= \lceil \lg \lceil CK \lg K \rceil \rceil + 2\lceil \lg \lceil CK' \lg K' \rceil \rceil \tag{19}\\
&\approx \lg K + \lg C + \lg\lg K + 2(\lg K' + \lg C + \lg K') \tag{20}\\
&\approx \lg\lg M + \lg C + \lg\lg\lg M + 2(\lg\lg\lg M + \lg C + \lg\lg\lg M) \tag{21}\\
&\approx (1 + 3\alpha + o(1)) \lg\lg M \tag{22}
\end{aligned}$$

which was what was wanted. ■

The error analysis this code depends on the prime number generator. We will be using a simple prime number generator based on the Miller-Rabin primality test.

6.3 Miller-Rabin analysis

Miller-Rabin is a well-known random primality test algorithm. Let $MR(n, k)$ be the distribution of Miller-Rabin algorithm on the prime candidate n with k repeats. Let $\mathcal{P}$ be the set of primes. Then, we have the following probabilities.

$$\begin{aligned}
\Pr[MR(n, k) \mid n \in \mathcal{P}] &= 1 \tag{23}\\
\Pr[MR(n, k) \mid n \notin \mathcal{P}] &\leq 4^{-k} \tag{24}
\end{aligned}$$

Consider the following random prime number genrator, $GMR(N, s, k)$. This algorithm, samples number uniformly and then checks if they are prime using the Miller-Rabin test. Then, N is the upperbound, s is the maximum number of attempts, and k is the number of repeats in the underlying Miller-Rabin test. We analyse the distribution of $GMR(N, s, k)$. Let n_i denote the random variable n in the i_{th} iteration.

Algorithm 4: $GMR(N, s, k)$

```
input  : positive integers $N, s, k$
output: A uniformly chosen odd prime less than or equal to $N$
for $i = 1 \to s$ do
    $n \leftarrow \{3, 5, \ldots, N\}$
    if $MR(n, k)$ then
        return $n$
    end
end
return $\perp$
```

$$\begin{aligned}
\Pr[GMR(M, s, k) = \perp] &= \Pr[MR(n_1, k) = \cdots = MR(n_s, k) = 0] \tag{25}\\
&= \prod_{i=1}^{s} \Pr[MR(n_i, k) = 0] \qquad \text{(Independence)} \tag{26}\\
&= \prod_{i=1}^{s} \Pr[MR(n_i, k) = 0 \mid n_i \notin P] \Pr[n_i \notin P] \tag{27}\\
&\leq \prod_{i=1}^{s} \left(1 - \frac{\pi(N)}{N}\right) \tag{28}\\
&= \left(1 - \frac{\pi(N)}{N}\right)^{s} \tag{29}\\
&\approx \left(1 - \frac{1}{\ln N}\right)^{s} \qquad \text{(PNT)} \tag{30}
\end{aligned}$$

If we bound this error probability with ϵ, then we get the following bound on s.

$$s \geq \frac{\ln \epsilon}{\ln\left(1 - \frac{1}{\ln N}\right)} \approx -\ln N \ln \epsilon$$

For sufficiently large N. The probability that the result of $GMR(N, s, k)$ is composite, given it is not $\perp$ is as follows.

$$\begin{aligned}
\Pr[GMR(N,s,k) \notin P \mid GMR(N,s,k) \neq \perp] &\leq \sum_{i=1}^{s} \Pr[MR(n_i,k) = 1, n_i \notin P] && (31)\\
&= \sum_{i=1}^{s} \Pr[MR(n_i,k) = 1 \mid n_i \notin P] \Pr[n_i \notin P] && (32)\\
&= \sum_{i=1}^{s} 4^{-k}\left(1 - \frac{\pi(N)}{N}\right) && (33)\\
&= s4^{-k}\left(1 - \frac{\pi(N)}{N}\right) && (34)\\
&\approx s4^{-k}\left(1 - \frac{1}{\ln N}\right) \qquad \text{(PNT)} && (35)
\end{aligned}$$

If we bound this error probability with δ, then we get the following bound on s.

$$s \leq \left(1 - \frac{1}{\ln N}\right)^{-1} 4^k \delta \approx 4^k \delta \tag{36}$$

Let $\epsilon = e^{-l}$ and $\delta = 2^{-q}$ with $l, q \geq 0$. Then,

$$l \ln N \leq s \leq 2^{2k-q} \tag{37}$$

Note that, $s = l \ln N$ and $k = \dfrac{\lg(l/e) + \lg\lg N + q}{2}$ satisfies both inequalities.

6.4 Proof of modified 3-step algorithm

Theorem 9. *The modified 3-step algorithm produces a $(n = n(M), M, \lambda_1 = 0, \lambda_2 = \lambda_2(M, C))$ identification code – per **??**– such that $\lambda_2 \to 0$ as $M \to \infty$. Moreover, this coding scheme is optimal.*

$$\lim_{M\to\infty} \frac{\lg\lg M}{n(M)} = 1 \tag{38}$$

We employ the same techniques used by Ahlswede by first calculating the following probability. Suppose $m, \hat{m} \in \mathcal{M} = \{1, 2, \ldots, M\}$ and $p \leftarrow GMR(N, s, k)$ such that the $\Pr[p = \perp] = 0$ [1] and $\Pr[p \notin \mathcal{P}] \leq \delta$ then

$$\begin{aligned}
\Pr[\phi_p(m) = \phi_p(\hat{m}) \mid m \neq \hat{m}] &= \Pr[\phi_p(m) = \phi_p(\hat{m}) \mid m \neq \hat{m}, p \in \mathcal{P}] \Pr[p \in \mathcal{P} \mid m \neq \hat{m}] && (39)\\
&\quad + \Pr[\phi_p(m) = \phi_p(\hat{m}) \mid m \neq \hat{m}, p \notin \mathcal{P}] \Pr[p \notin \mathcal{P} \mid m \neq \hat{m}] && (40)\\
&\leq \frac{\lg M}{\pi(N)} \Pr[p \in \mathcal{P}] + \Pr[p \notin \mathcal{P}] && (41)\\
&\leq \frac{\lg M}{\pi(N)} + \delta && (42)
\end{aligned}$$

Proof. Let $m, \hat{m} \in \mathcal{M} = \{1, 2, \ldots, M\}$ be given. Let the probable prime $k \leftarrow GMR(\lceil CK \lg K\rceil, s_K, r_K)$ and $l \leftarrow GMR(\lceil CK' \lg K'\rceil, s_{K'}, r_{K'})$ with $K = \lceil \lg M\rceil$ and $K' = \lceil \lg K\rceil$. The parameters $r_K, s_K, r_{K'}$, and $s_{K'}$ are set such that

$$\begin{aligned}
\Pr[k = \perp] = \Pr[l = \perp] = 0 && (43)\\
\Pr[k \notin P], \Pr[l \notin P] \leq \delta && (44)
\end{aligned}$$

[1] This may be achieved by either selecting a small prime number or a random number instead of outputing $\perp$. Since we have shown that this probability can be as small as we'd like, the first method shall not affect our uniform distribution considerably.

The probability of the second kind error is given as follow

$$\begin{aligned}
P_{e,2}(m,\hat{m}) &= \Pr[\phi_l(\phi_k(m)) = \phi_l(\phi_k(\hat{m})) \mid m \neq \hat{m}] && (45)\\
&= \Pr[\phi_l(\phi_k(m)) = \phi_l(\phi_k(\hat{m})) \mid \phi_k(m) = \phi_k(\hat{m}), m \neq \hat{m}] \Pr[\phi_k(m) = \phi_k(\hat{m}) \mid m \neq \hat{m}] && (46)\\
&\quad + \Pr[\phi_l(\phi_k(m)) = \phi_l(\phi_k(\hat{m})) \mid \phi_k(m) \neq \phi_k(\hat{m}), m \neq \hat{m}] \Pr[\phi_k(m) \neq \phi_k(\hat{m}) \mid m \neq \hat{m}] && (47)\\
&\leq \Pr[\phi_k(m) = \phi_k(\hat{m}) \mid m \neq \hat{m}] + \Pr[\phi_l(\phi_k(m)) = \phi_l(\phi_k(\hat{m})) \mid \phi_k(m) \neq \phi_k(\hat{m})] && (48)\\
&\leq \frac{\lg M}{\pi(\lceil CK \lg K \rceil)} + \frac{\lg \lceil CK \lg K \rceil}{\pi(CK' \lg K')} + 2\delta && (49)\\
&\approx \frac{\lg(M) \ln(CK \lg K)}{CK \lg K} + \frac{\lg(CK \lg K) \ln(CK' \lg K')}{CK' \lg K'} + 2\delta \qquad \text{(PNT)} && (50)\\
&\approx \frac{\lg K + \lg C + \lg \lg K}{C \lg K} \ln 2 + \frac{(K' + \lg K' + \lg C)(\lg K' + \lg \lg K' + C)}{CK' \lg K'} \ln 2 + 2\delta && (51)\\
&= \frac{2 \ln 2}{C} + 2\delta + o(1) && (52)\\
&= \frac{2 \ln 2}{(\lg M)^{\alpha}} + 2\delta + o(1) && (53)
\end{aligned}$$

Since δ can be chosen as small as possible, then $\lambda_2 \to 0$ as $M \to \infty$. ■