Nibble Packet Shredding Protocol // attempt at c++

```
struct NPSP_Header {
    Nibble version;
    Nibble n;   // using reed Solomon code (n,k) over
    Nibble k;   // GF(2^q)
    Nibble packet_number;

    Doublet sequence_number;
}
int s = 0; // Global sequence number
void encode(Message m, Target t, Nibble n, Nibble k) {
    int l = length(m) * sizeof(Nibble);
    int L = l / k; //round up
    Nibble chunked_message[k][L]; //stack
    par for(int i=0; i<k; i++) {
        par for(int j=0; j<L; j++) {
            if(i*k+j < l) {
                chunked_message[i][j] = m[i*k+j];
            }else{
                chunked_message[i][j] = 0;
            }
        }
    }
}
```

```
Nibble expanded_msg[L][n];
par for(int i=0; i<L; i++) {
    expanded_msg[i] = Evaluate_over_GF(16)(Chunked_msg[i], [0...n]);
}
NPSP_Header h;
h.version = 0; h.n = n; h.k = k; h.sequence_number = s; s++;
Nibble prepped_msg[n][L] = transpose(expanded_msg);
for(int i=0; i<n; i++) {
    h.packet_number = i;
    send(Ch: prepped_msg, t); //concat operator :
```

parallelizing requires many headers

Scanned with CamScanner

Nibble counter[s]; packet data[s][n]; Nibble n; Nibble k; Boolean flags[s];

void handler(packet p)

(h, p) = unpack(p);

assert(h.version == 0);

assert(h.n == h.k+2); // different code for last packet

assert(h.n == n);

assert(h.k == k);

if(data[h.sequence_number][h.packet_number] != null) {

delete p; return;

} if (flags[h.sequence]) { delete p; return; }

data[h.sequence_number][h.packet_number] = p;

counter[h.sequence_number]++;

if (counter[h.sequence_number] >= k+2 && !flags[h.sequence_number])

{

(flags[h.sequence_number], msg) = decode(data[h.sequence_number], n, k);

if (flags[h.sequence_number]) {

send(msg);

}

for (int j; j < n; j++) {

delete data[h.sequence_number][j];

counter[h.sequence_number] = 0;

}

}

// write decode

// find library for GF(16)

// reset flags.

// buffer of $2^{16} \times 16 = 1MB$

// failure to decode at counter == n case

// failure to decode timeout case

Scanned with CamScanner

Doublef Packet Shredding Protocol

```
Struct DPSP_Header{
    uint8_t  version;
    uint8_t  reserved;
    uint16_t packet_id;
    uint16_t message_id;
    uint16_t stream_id;
}
```

declare message_counter,
n_list key_list
k_list target_list

Get notation & objects vs Array notation & arrays

```
Struct DPSP_footer {
    ~~Byte~~ ByteArray[256] data
}
```

```
void encode (Message m, ~~Target t~~, uint16_t stream_id) {
    ~~acquire~~ (int n = n_List[stream_id]; int k = k_List[stream_id];
    acquire_lock( messageCounter[streamId] );
    int msg_id = messageCounter[streamId]++;
    release_lock(messageCounter[streamId]);
    ~~uint16_t chunked_msg[n_list[stream_id]]~~
    int l = length(m); // syntax wrong
    int L = l/m + (0==l%m)? 0 : 1;
    uint16_t chunked_msg[L][K];
    parfor (int i=0; i < L; i++) {
        parfor (int j=0; j<k; j++) {
            if (i*k+j < l) {
                chunked_msg[i][j] = m[i*k+j]
            }else{
                chunked_msg[i][j] = 0
            }
        }
    }
```

Scanned with CamScanner

void ...

```
uint16_t evaluation_pts[n];
parfor (int i; i<n; i++) {
    evaluation_pts[i] = i;
}
uint16_t expanded_msg[L][n];
parfor (int i; i<L; i++) {
    expanded_msg[i] = evaluation_over_GF(65536)(
        chunked_msg[i], evaluation_pts);
}
uint16_t prepped_msg[n][L] = transpose(expanded_msg);
parfor (int i=0; i<n; i++) {
    DPSP_header h;
    h.version = 0; h.reserved = 0; h.packet_id = i;
    h.message_id = msg_id; h.stream_id = stream_id;
    DSPS_footer f;
    f.data = sign(h:prepped_msg[i], key_list[stream_id]);
    send(h:prepped_msg[i]:f, target_list[stream_id]);
    // note : is the concat operater
}
```

~~void hard~~
~~uint16_t *counters = (array on stream_id) // for some thing fancy~~

Declare a few thing

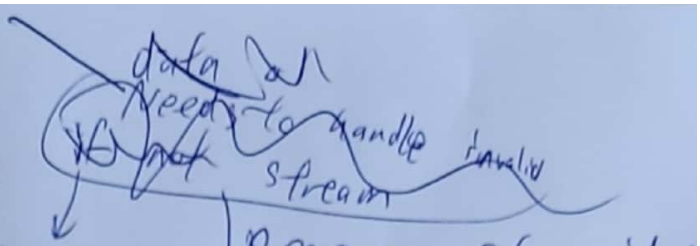
key_list
n_list
k_list

~~stream~~ ~~buffers~~ mailroom

data object
need better name

Sending
code

Scanned with CamScanner



Reserve stream_id = 0 for failed unpacking

```
void handler (packet p)
(h: packet_cur, f) = unpack (p);
verify_signature (key_list [h.stream_id]
if (key_list [h.stream_id] == null) {
    delete p; return;
}
if (!verify_signature (key_list [h.stream_id], h: packet_cur)) {
    delete p; return;
}
if (mailroom [h.stream_id] [h.message_id] == null) {delete p return}
if (mailroom [h.stream_id] [h.message_id] [h.packet_id] != null) {
    delete p; return;
}
acquire_lock (mailroom [h.stream_id] [h.message_id]);
mailroom [h.stream_id] [h.message_id].counter ++
mailroom [h.stream_id] [h.message_id] [h.packet_id] = packet_cur
if (mailroom [h.stream_id] [h.message_id].counter ++ >= k) {
    release_lock (      ); return;
}
else
encrypted_msg = mailroom [h.stream_id] [h.message_id].deepcopy();
release_lock (      );
send (decoder
send (decode (encrypted_msg));
}
```

Scanned with CamScanner