Let $e_A(x) = d_A(x) = x\text{^}748$ and $e_B(x) = d_B(x) = x\text{^}2455$ for some x, where ^ operator is xor, like in C language. Then, for example for $w = 5689$:
$e_A(w) = 5689\text{^}748 = 5333$
$e_B(e_A(w)) = e_B(5333) = 5333\text{^}2455 = 7490$
$d_A(e_B(e_A(w))) = d_A(e_B(5333)) = d_A(7490) = 7490\text{^}748 = 8110$
$d_B(d_A(e_B(e_A(w)))) = d_B(d_A(e_B(5333))) = d_B(d_A(7490)) = d_B(8110) = 8110\text{^}2455 = 5689$

No, I have no idea about any modern commutative cryptosystem.

The key is computed as $K = q^{xy} \bmod p$, so the value is

$K = (2^{15593477}) \bmod 9887 = (2^{5891 \cdot 2647}) \bmod 9887 = \left((2^{5891} \bmod 9887)^{2647}\right) \bmod 9887 =$
$= (3621^{2647}) \bmod 9887 = 1540$

No, it doesn't generate $\mathbb{Z}_p$, e. g. 5 is not generated by any power of 2 (generated elements repeat after 2^{4943}).
Order of that group is 4943, it's a prime, so security in this way is preserved.

The first thing, we must do is computing W^{-1}. 173^{-1} in $(\mathbb{Z}_{2048}, \cdot)$ is 805 ($173 \cdot 805 \bmod 2048 = 1 = \mathbf{1}$ in that group). Now, we can compute c' for each number c in cryptotext as $c' = cW^{-1} = c \cdot 805$ (everything mod 2048). And that's all what we need. Now we are able to solve modified knapsack problem for computed c's by getting the highest number from X and trying if it fits in that number. If yes, we subtract it and get predecessor of the highest. Otherwise – if not – we only don't subtract, but try the smaller one. For each, that fits, we write 1, otherwise 0. When we read the resulting 0s and 1s from the end, we have the corresponding plaintext. In math words:

(a) $W = 173$, for each $a \in \mathbb{Z}_{2048}$ try if $aW = 1 \,(\mathrm{mod}\, 2048)$, if yes, stop and return it (we know from algebra, that if it exists, it exists only once, any other occurence is equivalent to it). The result is, as mentioned before, $173^{-1} = 805$.

(b) $1786' = 1786 \cdot 805 \bmod 2048 = 34$
$5596' = 1228$

(c)

number	34	56	93	195	432	943
34	1	0	0	0	0	0
1228	1	1	0	1	0	1

(d) $d(1786) = (100000)_2 = 32$
$d(5596) = (110101)_2 = 53$

Decryption of the next message:

cryptotext	1008	1347	2463	5075	4257	3100	3728	1347
cryptotext'	432	943	251	1003	581	1036	720	943
binary plaintext	000010	000001	010100	001111	011010	001001	001110	000001
decimal plaintext	2	1	20	15	26	9	14	1
plaintext	B	A	T	O	Z	I	N	A

The initial password number is 32478271, the 9th round password is '86 d2 5f f2 41 47 8c a7 2c c9 59 9f aa f2 5f 0d' denoted in hexadecimal numbers. I get it with this C code:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <openssl/md5.h>

static unsigned char res[] = { 0x6f, 0x05, 0x09, 0x93, 0x17, 0x25, 0x59, 0x66,
        0x90, 0xcc, 0x2a, 0x67, 0x33, 0x79, 0xd1, 0xb1 };

void comb(const unsigned int c, const char pis[20])
{
        unsigned char x;

        if (c == 0) {
                unsigned char *m = MD5((unsigned char *)pis, 8, NULL);
                for (x = 0; x < 9; x++)
                        m = MD5(m, 16, NULL);

                for (x = 0; x < 16; x++)
                        if (m[x] != res[x])
                                return;

                printf("I got it: %s\n", pis);
                exit(0);
        } else
                for (x = '0'; x <= '9'; x++) {
                        char pom[20];
                        unsigned int a;

                        strcpy(pom, pis);
                        a = strlen(pom);
                        pom[a] = x;
                        pom[a + 1] = 0;
                        comb(c - 1, pom);
                }
}

int main()
{
        comb(8, "");

        return 0;
}
```

in approx. 10 mins followed by:

```
$ echo -n 32478271|openssl md5 -binary|openssl md5 -binary| \
> openssl md5 -binary|openssl md5 -binary|openssl md5 -binary| \
> openssl md5 -binary|openssl md5 -binary|openssl md5 -binary|openssl md5
86d25ff241478ca72cc9599faaf25f0d
```

which writes wanted MD5 result.
FYI the 'else' case in the code generates possibilities of starting password, 'if' case then hash it 10 times and compare it to the first different byte. If codes are equivalent it writes 'I got it: number', where number is the initial password and terminates the execution.

The first problem is, that n should be result of multiplication of some 2 prime numbers. We can factorize the number as $1159529 = 1 \cdot 1159529 = 7 \cdot 165747 = 151 \cdot 7679 = 1057 \cdot 1097$, but at least one of each pair is not a prime. Next e is not the inverse for d in none of that 3 cases, what is one of conditions, which have to be satisfied to not allowing the attacker to factorize the number quickly.

So, we can choose some plaintext m and calculate ciphertext $c = m^2 \mod n$. Next we can decrypt c as follows: we need to know m_1 (it may be $m \neq m_1$, because of modulo operations; the 2 same numbers may give us the same result). We can compute

$$m^2 - m_1^2 = (m - m_1)(m + m_1) = 0 \mod n$$

and we can easily obtain a factor of n (this problem isn't only in NP, but also in P).

Pretty easy, isn't it? Yes, indeed. When I was looking at the public key – the G' matrix – I noticed, that there is an identity submatrix and hence (with some high probability) P and S are also identity matrices. Then I realized, that $G = G'$ (generator matrix is equivalent to public matrix) and it was sufficient to decode the ciphertext with that G. In fact it is only error-correcting code defined by generator matrix, so it was not so hard to decode it by generating check matrix and computing syndrome for each 15-tuple with decoding after it. Next, 11-tuples (G is 11x15, redundancy is covered in 4 bits) were catenated and seperated in 8-tuples – 1 byte. In the next step it was pretty easy to paraphrase to convert binary string into letters. As you noticed, the first sentence is the resulting words even with punctuation and spaces.

This is known as Hastad's broadcast attack. Ok, let's get in. What do we have? We have ciphertexts c_A, c_B, ..., c_E and we know, that $e = 5$. We also know, that the gcd of ns is 1 and $n_X \neq n_Y \, \forall X \neq Y$. We can write:

$$c_A = m^5 \mod n_A, \; c_B = m^5 \mod n_B, \; \ldots, \; c_E = m^5 \mod n_E$$

where m is plaintext. By applying Chinese remainder theorem to c_A, c_B, c_C, c_D, c_E, we get some $c' \in \mathbb{Z}_{n_X}$ (say $n_X = n_A n_B n_C n_D n_E$) satisfying $c' = m^5 \mod n_X$. We have $m^5 < n_X$, because $m < min(n_A, \ldots, n_E)$. Now $c' = m^5$ and it's much easier to compute it than NP-hard factorizating numbers.

It is possible to do that when number of keys/peoples is greater than e.