TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN
KHOA KỸ THUẬT MÁY TÍNH

ĐỒ ÁN CUỐI KÌ
MÔN CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT
Đề tài: Phá mã mật khẩu được mã hóa bằng subset sum

SV thực hiện: Đinh Thành Phát
MSSV: 21520083
Lớp: ATTN2021
Người hướng dẫn:

- Nguyễn Thanh Sơn
- Nguyễn Đức Vũ

I, GIỚI THIỆU

#	8-char key	40-bit key
0	4szcirzz	1111010010110010001001000100011100111001
1	ibsalkc2	0100000001100100000001011010100001011100
2	usnfujjz	1010010010011010010110100010010100111001
3	bu4mmejr	0000110100111100110001100001000100110001
4	5bd5qclw	1111100001000111111110000000100101110110
5	5vkmuyjo	1111110101010100110010100110000100101110
6	v521cfzw	1010111111111001101100010001011100110110
7	ty12gjq2	1001111000110111110000110010011000011100
8	35b5hs0a	1110111111000011111100111100101101000000
9	0q0w2nna	1101010000110101011011100011010110100000
10	uwnzdxew	1010010110011011100100011101110010010110
11	xmjjhv3s	1011101100010010100100111101011110110010
12	lb3ffann	0101100001111010010100101000000110101101
13	za5lkafk	1100100000111110101101010000000010101010
14	dniohm4x	0001101101010000111000111011001111010111
15	20uqpma2	1110011010101001000001111011000000011100
16	4ac0ist0	1111000000000101101001000100101001111010
17	2c24x3h2	1110000010111001111010111111010011111100
18	c4u2phiq	0001011110101001110001111001110100010000
19	zdvpqfb0	1100100011101010111110000001010000111010
20	wiszzamb	1011001000100101100111001000000110000001
21	0nbn5c0v	1101001101000010110111111000101101010101
22	aipxjpv4	0000001000011111011101001011111010111110
23	iphcrucy	0100001111001110001010001101000001011000
24	3leopdkz	1110101011001000111001111000110101011001
25	3wzkljly	1110110110110010101001011010010101111000
26	robtlkyw	1000101110000011001101011010101100010110
27	h0bjkm2m	0011111010000010100101010011001110001100
28	nniflpbf	0110101101010000010101011011110000100101
29	wvo3no2d	1011010101011101110101101011101110000011
30	kg4ts5y2	0101000110111101001110010111111100011100
31	b12qgbzu	0000111011111001000000110000011100110100
32	wwahbmfq	1011010110000000011100001011000010110000
33	fu15rfbv	0010110100110111111110001001010000110101
34	hv2as24l	0011110101111000000010010111001111001011
35	vw3dilsf	1010110110111010001101000010111001000101
36	1dcqok04	1101100011000101000001110010101101011110

37	owryddqr	01110101101000111000000110001110000010001
38	kvjwuzgn	01010101010100110110101001100100110011101
39	ieae2oph	01000001000000000100111000111001111001111

Bảng 1. Ví dụ về bảng T cho mật khẩu có độ dài là 8 sử dụng bảng chữ cái 32 ký tự.

Cho 1 bảng T gồm N dãy nhị phân N-bit. Đồng thời chúng ta được cung cấp bảng chữ cái gồm 32 kí tự " abcdefghijklmnopqrstuvwxyz012345". Hệ mã hóa trên sử dụng bảng T để mã hóa chuỗi kí tự(tạm gọi là password) thành 1 chuỗi khác dựa trên ý tưởng bài toán subset sum. Chúng ta sẽ tìm hiểu thông quá ví dụ sau.

Đầu tiên về quy ước biến đổi, ta sẽ biến đổi các kí tự trong bảng chữ cái trên thành số nhị phân 5 bit, tức là chữ cái "a" sẽ tương ứng với số 00000, "b" tương ứng với "00001", "c" tương ứng với "00010" … Như vậy, nếu chuỗi password có độ dài là C kí tự thì sau khi biến đổi sang nhị phân sẽ có độ dài là N = 5C. Tiếp theo, giả sử chuỗi được cho là "aaaaaaaa" , sau khi biến đổi sẽ ra "0000000000000000000000000000000000000000", vì chuỗi biến đổi không tồn tại bit 1 nên sau khi mã hóa sẽ ra chuỗi ban đầu là "aaaaaaaa" . Tiếp theo là chuỗi "bbbbbbbb", biến đổi sang dạng nhị phân sẽ là "0001000010000100001000010000100001000010" . Ta thấy các bit 1 tương úng có trong chuỗi là bit thứ 4,9,14,19,24,29,34,39(tính từ trái sang). Chúng ta sẽ lấy các chuỗi T[i] tương úng có trong bảng T và cộng chúng lại rồi đem chia lấy dư cho 2^{40}:

#	8-char key	40-bit key
4	5bd5qclw	1111100001000111111110000000100101110110
9	0q0w2nna	1101010000110101011011100011010110100000
14	dniohm4x	0001101101010000111000111011001111010111
19	zdvpqfb0	1100100011101010111110000001010000111010
24	3leopdkz	1110101011001000111001111000110101011001
29	wvo3no2d	1011010101011101110101101011101110000011
34	hv2as24l	0011110101111000000010010111001111001011
39	ieae2oph	01000001000000000100111000111001111001111
	z3mfp5nv	10011001111010110000101011111111110110110101

Như vậy, sử dụng bảng T như đề cập trong Bảng 1, mật khẩu "bbbbbbbb" sẽ được mã hóa thành "z3mfp5nv".

Yêu cầu đặt ra:

1. Viết chương trình đầu nhận vào mật khẩu đã được mã hóa gồm C ký tự (C có giá trị tối đa là 12) sử dụng bảng chữ cái 32 ký tự theo phương pháp mô tả ở trên. Ngoài ra, chương trình còn được cho biết trước bảng T gồm N=5*C dòng, mỗi dòng là một N-bit key nào đó. Phân tích độ phức tạp của chương trình theo C. Đầu ra của chương trình là mật khẩu trước khi được mã hóa.
2. Tồn tại trường hợp hai mật khẩu có cùng mật mã được mã hóa. Hãy trình bày một hướng giải quyết để tạo bảng T sao cho không còn tồn tại trường hợp này nữa.

II, GIẢI QUYẾT VẤN ĐỀ

1, Yêu cầu thứ 1

- Cho 1 xâu với các kí tự thuộc bảng “abcdefghijklmnopqrstuvwxyz012345", với mỗi kí tự trong xâu viết chúng thành dãy 5 bit liền kề(vd ‘a’ tương ứng với 00000, ‘b’ tương ứng với ‘00001’, …)
- Sau khi được xâu nhị phân mới từ xâu ban đầu đã cho, lúc này ta xét tới bảng T.
- Với mỗi bit thứ i là 1 trong xâu nhị phân trên, ta cộng với dãy nhị phân tương ứng trong bảng T
- Sau khi được kết quả cuối cùng, đem kết quả mod cho 2^(n + 1), rồi mã hóa ngược lại xâu nhị phân(chuyển từng 5 bit trên xâu nhị phân thành các kí tự), cuối cùng ta thu được xâu mã hóa của xâu ban đầu.
- Có thể nhận ra rằng , quá trình biến đổi chuỗi password thành 1 chuỗi mã hóa dựa trên ý tưởng của bài toán subset sum, cho nên nếu đưa ra được hướng giải quyết của bài toán subset sum, chúng ta hoàn toàn có thể khôi phục chuỗi ban đầu dựa trên chuỗi đã cho.
- Tuy nhiên bài toán subset sum là 1 bài toán np complete (tức là không tồn tài lời giải trong thười gian đa thức)
- Phương án để làm là duyệt trâu các tổ hợp để tìm ra subset có tổng bằng với yêu cầu đã cho. Độ phức tạp thuật thuật toán là $O(2^n)$.
- Có thể sử dụng kĩ thuật meet in the middle để giảm thời gian chạy thuật toán. Độ phức tạp mới sẽ là $O(2^{n/2})$.
  Chứng minh:

Giải thích ý tưởng meet in the middle:

- Gọi S là tổng yêu cầu của bài toán, A là tập các phần tử(A = { a1,a2,…an }), B là subset lời giải của bài toán, ta có:

Dễ dàng chứng minh được B = B1 + B2, với B1 B2 là subset con của B

Sum(B1) + Sum(B2) = S

⇨ Sum(B1) = S – Sum(B2)

Nhận xét: B1 và B2 là hai tập hoàn toàn độc lập với nhau, nên có thể tính trước tập B1, sau đó đi tìm tập B2.

Ý tưởng là chia tập A thành hai phần bằng nhau để giảm độ phức tạp duyệt trâu. Sau đó, sinh nhị phân trên tập A1,A2(sinh nhị phân hai tập riêng biệt nên độ phức tạp là $O(2^{n/2})$), với mỗi trạng thái nhị phân trên tập A2, ta được 1 giá trị mới là S – sum(state), khi này có thể lưu chúng trong ctdl như set để giảm độ phức tạp khi tìm kiếm. Cuối cùng với mỗi trạng thái A1, ta tìm xem tổng của state liệu có tồn tại ở <A2> không (thông qua set) với đpt $O(\log 2^{n/2})$.

Độ phức tạp của thuật toán : $O(2^{n/2}.\log(2^{n/2}))$.

Tông quát lại các bước cần làm:

- Bước 1: Nhập dữ liệu
- Bước 2: Tạo mảng A từ bảng T
- Bước 3: Tạo mảng A1 là mảng sinh từ N/2 phần từ đầu tiên của mảng A, A2 là mảng sinh ra từ N/2 phần tử cuối cùng của mảng A(sử dụng phương pháp sinh nhị phân cho cả hai mảng A1,A2)
- Bước 4: Duyệt qua các giá trị trong A1, tạm gọi giá trị đang duyệt là F, tìm xem giá trị S – F(S ở đây là encrypted password) có tồn tại trong A2 hay không, nếu tồn tại, ghi nhận lại kết quả, nếu không, duyệt qua phần tử kế tiếp.
- Bước 5: In ra các giá trị có hash bằng với encrypted password.

Source code:

```
#include <bits/stdc++.h>
#define _for(i,a,b) for(int i=(a),_b=(b);i<_b;++i)
///made by d4rkn19ht
using namespace std;
typedef long long ll;
ll MOD;
typedef pair<ll,ll> pir;
#define fi first
#define se second

ll bin_to_num(string &a)
{
    ll num = 0;
    for(auto i:a)
    {
        num = (num<<1) + (i - '0');
    }
    return num;
}
ll char32_to_int(char a)
```

```
{
    if ('a' <= a && a<='z')
    {
        return a - 'a';
    }
    else
    {
        return a - '0' + 26;
    }
}
char int_to_char32(ll a)
{
    if (a>=26)  return char(a - 26 + '0');
    else return char(a + 'a');
}


bool valid_char(char a)
{
    if ('a'<=a && a<='z') return true;
    if ('0'<=a && a<='5') return true;
    return false;
}
ll string_to_num(string a)
{
    ll num = 0;
    for(auto i:a)
    {
        for(int j = 4; j>=0; --j)
        {
            num = num<<1;
            if (char32_to_int(i)&(1<<j)) num += 1;
        }
    }
    return num;
}
string num_to_string(ll a,int t)
{
    string b = "";
    _for(i,0,t/5)
    {
        int s = 0;
        _for(j,0,5)
        {
            s = s | ((a&1)<<j);
```

```
            a >>=1;
        }
        b += int_to_char32(s);
    }
    reverse(b.begin(), b.end());
    return b;
}
void nhap_du_lieu(vector<ll>& a)
{
 ///   ifstream fi;
///    fi.open("data.inp");
  ///  freopen("data.inp","r",stdin);
    int N;
    cin>>N;
    N = N;
    _for(i,0,N)
    {
        string bin;
        cin>>bin;
        ll num = bin_to_num(bin);
        a.push_back(num);
    }

}

void snp(int pos, ll sum,ll state, vector<ll>& numset, vector<pir>& gen)
{
    if (pos >= int(numset.size()))
        gen.push_back(pir(sum, state));
    else
    {
        snp(pos + 1, sum,state, numset, gen);
        snp(pos + 1, (sum + numset[pos])%MOD,state | (ll(1) << pos), numset,
gen);
    }
}
ll encoding_password(string password, vector<ll>& numset)
{
    ll num = 0;
    ll state = string_to_num(password);
    int N = numset.size();
    _for(i,0,N)
    {
        if (state & (ll(1)<<i))
            num += numset[i];
```

```
        num %= MOD;
    }
    num %= MOD;
    return num;
}
bool checking(string password, string encoded_password, vector<ll>& numset)
{
    if (num_to_string(encoding_password(password, numset), int(numset.size())) ==
encoded_password) return true;
    else return false;
}
void password_crzcking(string encoded_pass, vector<ll>& numset)
{
    /**
    cout<<"DEBUGGING MSG: bbbbbbbb = "<<string_to_num("bbbbbbbb")<<"\n";
    cout<<"DEBUGGING MSG: "<<string_to_num(encoded_pass)<<"\n";
    cout<<"DEBUGGING MSG: "<<encoding_password("bbbbbbbb",numset)<<"\n";
    cout<<"DEBUGGING MSG: h(\"bbbbbbbb\") =
"<<num_to_string(encoding_password("bbbbbbbb",numset), int(numset.size()))<<"\n";
    */
    /// change password from string to number
    ll enc_num = string_to_num(encoded_pass);

    MOD = ll(1) << int(numset.size());/// create MOD

    vector<ll> numset1, numset2;
    /// divide set into 2 equal-length sets
    int len = int(numset.size());
    _for(i,0,len/2)
        numset1.push_back(numset[i]);

    _for(i,len/2, len)
        numset2.push_back(numset[i]);

    /// generate 2^(len/2) subset sum
    vector<pir> gen1,gen2;
    snp(0, 0, 0,numset1, gen1);
    snp(0, 0, 0,numset2, gen2);

    ///sorting gen2
    sort(gen2.begin(), gen2.end());

    /// checking if password is valid
    if (len != int(encoded_pass.size())*5)
    {
```

```
        cout<<"Invalid password!!!!\n";
        return ;
    }
    for(auto i:encoded_pass)
        if (!valid_char(i))
        {
            cout<<"Invalid password!!!!\n";
            return ;
        }

    /// cracking
    vector<string> found_password;
    for(auto i:gen1)
    {
        ll p1 = (enc_num + MOD - i.fi)%MOD;
        int pos = lower_bound(gen2.begin(), gen2.end(), pir(p1, 0)) -
gen2.begin();
        if (pos == int(gen2.size())) continue;
        if ((gen2[pos].fi + i.fi)%MOD == enc_num) /// if found a way
        {
            ll state = (gen2[pos].se << int(numset2.size())) | i.se;
            string found_pass = num_to_string(state, int(numset.size()));
       ///     cout<<"Found password:"<<found_pass<<"\n";
            if (checking(found_pass, encoded_pass, numset))
            {
      ///          cout<<"Correct password!!!!\n";
     ///           cout<<"Recoverd password: "<<found_pass<<"\n";
      ///          return;
                 found_password.push_back(found_pass);
            }
      ///      else cout<<"Not right password!!!!\n";
        }
    }
    sort(found_password.begin(), found_password.end());
    for(auto i : found_password) cout<<i<<"\n";
  ///  cout<<"NOT FOUND!!\n";
}
int main()
{
    ios_base::sync_with_stdio();
    cout.tie();
    cin.tie();
    vector<ll> numset;
    nhap_du_lieu(numset);
 ///   cout<<"Finished reading data!!!\n";
```

```
    string enc_pass = "";
    MOD = ll(1) << int(numset.size());
    reverse(numset.begin(), numset.end());
///    cout<<"Nhap password de khoi phuc:\n";
    cin>>enc_pass;
    password_crzcking(enc_pass, numset);
    return 0;
}
```

T

Một số kết quả thu được trong quá trình thực nghiệm thuật toán:

Thứ tự	Độ dài	Thời gian phá mã	Bộ nhớ sử dụng
Test 1	C = 5	< 0.01s	<1KiB
Test 2	C = 6	<0.01s	<1KiB
Test 3	C = 8	0.8s	3096KiB
Test 4	C = 10	42s	6192KiB
Test 5	C = 12	Runtime Error	Runtime Error
Test 6	C = 8	39s	6205KiB

Có thể thấy rằng, trong source code trên, tôi đã sử dụng $O(2^{n/2})$ bộ nhớ, dẫn đến test số 5 bị lỗi(Runtime Error), tuy nhiên vẫn có cách tối ưu bộ nhớ hơn là sử dụng thuật toán Schroeppel and Shamir. Tôi sẽ để link để các bạn có thể tham khảo thêm.

2, Yêu cầu thứ 2

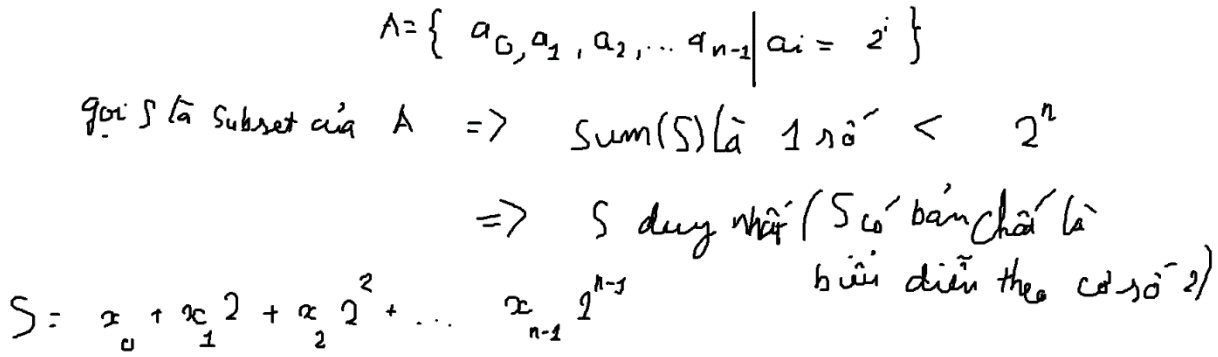
- Chúng ta được yêu cầu sinh ra 1 bảng T sao cho không tồn tại collision khi hash 2 chuỗi khác nhau.

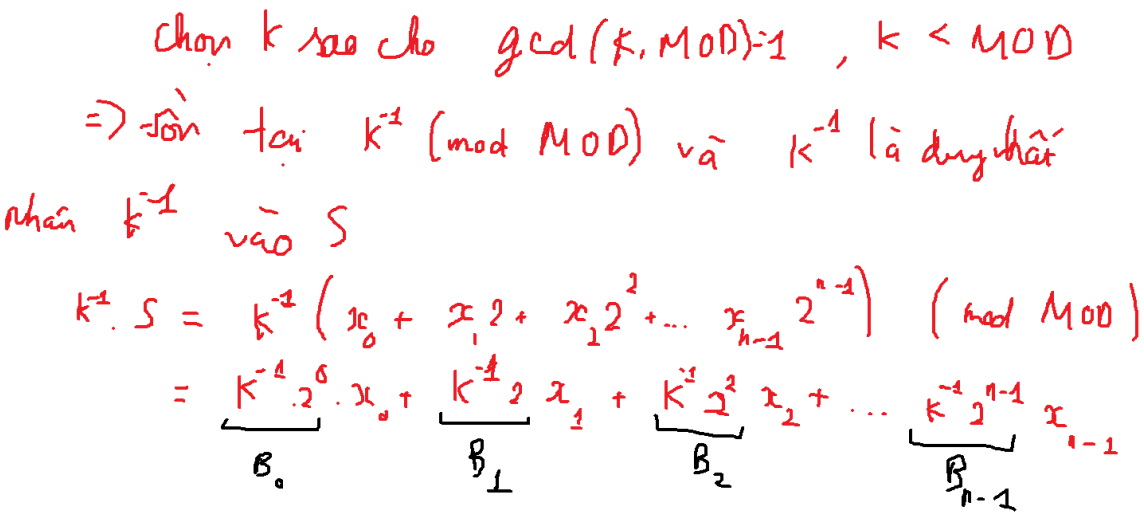
Ý tưởng xây dựng bảng

- Chọn 1 số k ngẫu nhiên sao cho gcd(k, MOD) = 1
- Tính inverse modulo của k (k^{-1}) mod MOD
- Xét tập A = {a0, a1, …a[n-1] | ai = 2^i}, nhân mỗi phần từ của A với k^{-1}.
- Gọi tập B là kết quả của bước 3(B = {b0, b1,b2, …,b[n-1] | b[i] = a[i]*k^{-1} % MOD})

- Chuyển các phần từ trong tập B về dạng nhị phân, khi này ta thu được bảng T sao cho ko tạo ra hash trùng.

Chứng minh:





Dễ nhận thấy rằng k^{-1} là duy nhất(mod MOD) nên suy ra tích $S.k^{-1}$ cũng là duy nhất(mod MOD)(có thể coi $F(S) = S*k^{-1}$ thì lúc này F(S) là 1 phép đẳng cấu). Đồng thời tổ hợp $f(X) = (x0,x1,\ldots xn-1) = k^{-1}(x0,x1,\ldots xn-1)k$ ($k.k^{-1} = 1$ mod MOD) cũng biểu diễn được cho mọi số nguyên < MOD => $g(X) = k^{-1}(x0,x1,\ldots xn-1)$ cũng biểu diễn được cho mọi số nguyên < MOD(f(X) và g(X) là song ánh)

Nhận xét: Tuy đã đạt được yêu cầu là sinh ra bảng T sao cho không tạo ra collision khi hash, tuy nhiên ý tưởng trên lại có thể dễ dàng bị hack do 1 số k bất kì khi nhân 2^i (mod 2^n) thực chất là phép toán shift left. Do đó rất dễ truy tìm được k và k^{-1} trong bảng T(k là số nguyên lẻ duy nhất trong bảng T vì $k*2^0 = k$). Tôi khuyến khích không nên sinh bảng T dựa theo thuật toán trên.

Source code:

```
#include <bits/stdc++.h>
#define _for(i,a,b) for(int i=(a),_b=(b);i<_b;++i)

using namespace std;
typedef long long ll;
typedef pair<ll,ll> pir;
ll mod;
#define fi first
#define se second

pir extend_euclid(ll a,ll b, ll c)
{
    ll x0 = 1, y0 = 0, r0 = a;
    ll x1 = 0, y1 = 1, r1 = b;
    while(r1 != c)
    {
        ll q = r0 / r1;
        ll r = r0%r1;
        ll x = x0 - x1*q, y = y0 - y1*q;
        x0 = x1;
        y0 = y1;
        x1 = x;
        y1 = y;
        r0 = r1;
        r1 = r;
    }
    return pir(x1, y1);
}
ll gcd(ll a,ll b)
{
    return (b == 0) ? a : gcd(b, a%b);
}
ll inverse_modulo(ll a, ll b)/// find the a^-1 of modulo b
{
    if (gcd(a, b) != 1)
    {
        cout<<"Cannot find the inverse modulo!!\n";
```

```
        exit(1);
    }
    pir ans = extend_euclid(a, b, 1);
    while(ans.fi  <0)
    {
        ans.fi += b;
    }
    return ans.fi%b;
}
///cannot multiply 2 40 bit numbers so i have to make a quick multiplier
ll nhan(ll a,ll b)
{
    if (b==0) return 0;
    else
    {
        ll p = nhan(a, b/2);
        p = p*2;
        p = p%mod;
        if (b%2) return (p + a)%mod;
        else return p;
    }
}
void gen_board()
{
    int N = 40;/// assume that the board will be 40 row, you can change it later
    ll a[100];
    _for(i,0,N)
    {
        a[i] = ll(1)<<i;
    }
    mod = ll(1)<<N;
    ll k = (((ll(rand())*ll(rand()))%mod)*ll(rand()))%mod;/// this is where
everything start
    if (k%2==0) k += 1;
    ll _k = inverse_modulo(k, mod);
    _for(i,0,N)
    {
        a[i] = nhan(a[i], _k);
    }
    cout<<N<<"\n";
    random_shuffle(a, a+N);
    _for(i,0,N)
    {
        for(int j = N-1; j>=0; --j)
        {
```

```
            cout<<((a[i]>>j)&1);
        }
        cout<<"\n";
    }
}
int main()
{
    ios_base::sync_with_stdio(0);
    cin.tie();
    cout.tie();
    srand(int(time(0)));
    /**test
    cout<<inverse_modulo(3, 16)<<"\n";
    cout<<inverse_modulo(11, 15)<<"\n";
    cout<<inverse_modulo(1001, 1021)<<"\n";
    cout<<inverse_modulo(1234567, 1000000007);
    */
    freopen("data.inp","w",stdout);
    gen_board();
    return 0;
}
```

III, TÀI LIỆU THAM KHẢO

- https://en.wikipedia.org/wiki/Subset_sum_problem
- https://arxiv.org/pdf/2010.08576v2.pdf **(Improving Schroeppel and Shamir's Algorithm for Subset Sum via Orthogonal Vectors)**
- https://www.youtube.com/watch?v=RUlIAc9Fqo4 **(Improving Schroeppel and Shamir's Algorithm for Subset Sum via Orthogonal Vectors Video)**

IV, LINK

https://github.com/sinkthemall/Do_an_cuoi_ki-pwd_cracking-