Contemporary Engineering Sciences, Vol. 10, 2017, no. 19, 921 - 931
HIKARI Ltd, www.m-hikari.com
https://doi.org/10.12988/ces.2017.79101

MSSP for 2-D Sets with Unknown Parameters and a Cryptographic Application

Nadav Voloch

Computer Science Department
The Open University of Israel, Israel

Abstract

The subset sum problem (SSP) can be simply described as: given a set of integers *A*, and an integer *s*, find a subset of items from *A* summing up to *s*. To this classic problem there are many variants; one is the multiple-subset problem (MSSP) in which there is a selection of items from a given set to several identical bins, having each bin capacity not exceeded, and the total weight of the items is maximized. Here a related different kind of problem is approached: given a set of sets $A = \{A_1, A_2, \ldots, A_n\}$, find an integer *s*, for which every subset of the given sets is summed up to, if such an integer exists. To this specific problem there are several parameters that are applicable: the size of each set-*m*, the number of sets-*n*, and the size of every integer (number of digits)-*d*. For equal sizes of these parameters, a cryptographic application is shown, in which the cipher text is the sequence of the given sets, the private keys are two of the parameters *m*, *n*, and *d*, and the encrypted plain text is the resulted sum *s*. As a variant of SSP, the problem is *NP-complete*, and for known private keys, a relatively efficient algorithm is given, based on dispensing non-relevant values of the possible sums.

Keywords: Subset sum problem; cryptography; multiple subset problem; NP-complete problems

1 Introduction

In computational complexity theory the subset sum problem (SSP) is a special case of the knapsack problem that has many variants, and is *NP-complete* ([1]).

The classic problem, as shown in [2], is described as follows: Let $A= \{a_1, a_2, \ldots, a_m\}$, $a_i \in \mathbb{N}$, and $s \in \mathbb{N}$, find $B \subseteq A$, for which $\sum_{ai \in B} a_i = s$. Meaning for a given a set of integers A, and an integer s, find a subset of items from A summing up to s. SSP, being *NP-complete*, has a naïve exponential time solution of cycling all of the possible subsets, and a slightly better ($O(2^{n/2})$) solution shown in [3]. The more efficient solutions are pseudo-polynomial dynamic programming solutions using a array with Boolean values that determines the existence of subsets adding up to the needed sum, as presented in [4] and [5], and efficient approximations algorithms such the ones shown in [6] and [7]. One variant of SSP is the multiple-subset problem (MSSP) in which there is a selection of items from a given set to several identical bins, having each bin capacity not exceeded, and the total weight of the items is maximized. Approximation algorithms for MSSP were shown in [8]. A special case of MSSP is the equal subset problem (ESS), in which given a set $A= \{a_1, a_2, \ldots, a_m\}$, $a_i \in \mathbb{N}$, and $s \in \mathbb{N}$, the problem is to decide whether two disjoint subsets A_1, $A_2 \subseteq A$ exist whose elements sum up to the same value, meaning $\sum_{ai \in A_1} a_i = s$ and $\sum_{ai \in A_2} a_i = s$. The problem and an approximation algorithm for solving it are presented in [9]. From ESS, several different problems and variants are derived as presented in [10] and an efficient approximation algorithm for it was shown in [11].
Here we address a different kind of problem, that resembles ESS, but handles a set of sets (rather than a single set of integers), and an unknown parameter of the sum (s), that is to be found. This problem has a cryptographic application that is shown in the following parts of this paper.

2 The problem, notations and examples

A problem arises when ESS is addressed in a different manner that is as follows: For a given family of sets of integers $A = \{A_1, A_2, \ldots, A_n\}$, find an integer s, such that for every set A_i some of its subsets is summed up to s, if such an integer exists. This problem resembles ESS by the fact that a similar sum has to be found in different subsets, but the constraints are different: first, a set of sets (2D array) is handled, meaning there is a given fixed number of items in every part of the problem (every set).

Second, the sum (s) is an unknown parameter- meaning it is to be found for the sets. This constraint gives an $O(n)$ complexity multiplier leap, since SSP has to be done for every possible s. An efficiency improvement for this leap will be shown in the following parts of this paper. Some small scale examples for multiple integrated sets are seen in table 1. As seen, implementing SSP for finding sums on multiple integrated sets is not injective, thus multiple results of s can appear (like in set C), or none (like in set B). As the number of sets (denoted n) rises, the probability of getting at least one result of s decreases, since there are less possibilities of sum equalities between the sets, and as the number of items in each sets (denoted m), the probability of getting at least one result of s increases, since there are more possibilities of sum equalities between the sets.

The problem has a cryptographic application for equal sizes of sets (*m*), from which we derive the number of sets-*n*, and an equal number of digits of every integer in the sets (denoted *d*). In this application, the cipher text is the sequence of the given sets, the private keys are two of the parameters *m*, *n*, and *d*, and the encrypted plain text is the resulted sum *s*. For this application, a singular result of *s* is needed, for an exact decryption of the cipher text.

Table 1
Small scale examples for multiple integrated sets SSP with unknown parameters

2D set	Set	Set items	*s*	summations
A	A_1	{22, 4, 23, 16}	49	22+23+4=49
	A_2	{8, 3, 17, 21}	49	17+21+8+3=49
	A_3	{8, 13, 9, 19}	49	13+9+8+19=49
B	B_1	{22, 3, 20, 15}	None	None
	B_2	{5, 1, 17, 21}	None	None
	B_3	{8, 10, 7, 19}	None	None
	B_4	{23, 5, 26, 19, 4}	None	None
C	C_1	{8, 15, 11, 9, 1}	10; 21	9+1=10; 9+1+11=21
	C_2	{13, 2, 7, 1}	10; 21	7+2+1=10; 7+1+13=21
	C_3	{18, 11, 10,19}	10; 21	10=10; 10+11=21

3 The algorithm for SSP with multiple integrated sets

3.1 Bounds for the sums

Finding *s* for the multiple integrated sets SSP problem can be done in a naïve manner, in which we run SSP on the Koiliaris and Xu algorithm (mentioned above in [5], denoted here as KXSSP, that is $O(s\sqrt{n})$), on all of the possible options of *s*. Here a question arises: what are all of the possible options of *s*? The answer to this question gives us accurate bounds that improve the complexity of the algorithm, and make it a bit more efficient.

The lower bound of *s* (denoted s_{min}) is the maximal minimum item of a set A_i. This bound is set from the SSP trivial constraint of not having an empty subset, meaning the minimal amount of items in a subset is 1. For every set A_i, we take the minimum item, and then compare all of the minimums to find which one is the

maximal one of them (*max (min (A_i)*), since a smaller number from any minimum item cannot be a sum of the A_i including it, as shown in proposition 1:

Proposition 1:

$$s_{min} \geq max\ (min\ (A_i))$$

Proof:

Proof by contradiction: let us falsely assume $s_{min} < max\ (min\ (A_i))$, meaning there exists such a $s_{min} < min\ (A_i)$, for a set A_i. Meaning that for some $A_{ij,}$ where $1 \leq j \leq |A_i|$, $A_{ij} \leq s_{min}$, hence $A_{ij} < min\ (A_i)$, in contradiction of the meaning of $min\ (A_i)$, thus inferring necessarily that $\boldsymbol{s_{min} \geq max\ (min\ (A_i))}$.

As an example we can see the minimums of set *A* in table 1:

- $min\ (A_1) = 4$
- $min\ (A_2) = 3$
- $min\ (A_3) = 8$

➔ $s_{min} = max\ (min\ (A_i)) = 8$

The upper bound of *s* (denoted s_{max}) is the minimum item-sum of a set *Ai*. This bound is by trivial constraint of a subset $B \subseteq A_i$, meaning the maximal subset of a set is the set itself, hence the sum $\sum A_{ij}$ is the maximal sum for any subset. For every set A_i, we take the set's sum, and then compare all of the sums to find which one is the minimal one of them ($min\ (\sum A_{ij})$) , where $1 \leq j \leq |A_i|$, since a bigger number from any subset's sum cannot be a sum of the A_i including it, as shown in proposition 2:

Proposition 2:

$$\boldsymbol{s_{max} \leq min\ (\sum A_{ij})}\ \text{, where } 1 \leq j \leq |A_i|$$

Proof:

Proof by contradiction: let us falsely assume $s_{max} > min\ (\sum A_{ij})$, meaning there exists such a $s_{max} > \sum A_{ij}$, for a set A_i. Meaning that for some A_i there exists a subset *B*, for which $\sum_{ai \in B} a_i > \sum A_{ij}$. This stands in contradiction of the constraint of a subset $B \subseteq A_i$, thus inferring necessarily that $\boldsymbol{s_{max} \leq min\ (\sum A_{ij})}$.

As an example we can see the sums of set *A* in table 1:

- $\sum A_{1j}$, where $1 \leq j \leq |A_1| = 22 + 4 + 23 + 16 = 65$
- $\sum A_{2j}$, where $1 \leq j \leq |A_2| = 8 + 3 + 17 + 21 = 49$
- $\sum A_{3j}$, where $1 \leq j \leq |A_3| = 8 + 13 + 9 + 19 = 49$

➔ $s_{max} = min\ (\sum A_{ij}) = 49$

3.2 The algorithm

After formulating the bounds, we can now describe the process of solving the SSP on multiple integrated sets with unknown sum, by using KXSSP on every possible s in the range of $s_{\min} \leq s \leq s_{max}$.
The algorithm is as follows:

SSP on multiple integrated sets with unknown sum (2D set A):

1. $s_{\min} \leftarrow max\ (min\ (A_i))$
2. $s_{\max} \leftarrow min\ (\sum A_{ij})$, where $1 \leq j \leq |A_i|$
3. list $S \leftarrow \{\}$
4. for $s_i \leftarrow s_{\min}$ to $s_i \leftarrow s_{\max}$ do:
   4.1 *counter* ← 0
   4.2 for A_1 to A_n do:
      4.2.1 if $KXSSP(A_i, s_i) = true$:
         4.2.1.1 *counter* ← *counter*+1
   4.3 if *counter* = *n:*
      4.3.1 $S_i \leftarrow s_i$
5. return S

3.3 Explanation and complexity analysis

Steps 1 and 2 are finding the bounds for s as described in section 3.1, the complexity of these steps (meaning n as the total number of items in the 2D set) is $O(n)$, for scanning all of the items in A, and finding the minimums and the sums. Step 3 is initializing a list, in which all of the sums that are achievable by subsets in each set will be stored (e.g. 10 and 21 in set C from table 1, or 49 in set A from table 1). This is the list that is the result of the algorithm, the complexity of this step is $O(1)$. Step 4 is the loop that checks all of the possible sums on A. In step 4.1 we initialize a counter. This counter will advance in case we find a subset that equals s_i in any of the sets A_i; this is done in step 4.2 where we run KXSSP to find if such a subset exists. In step 4.3 the counter is checked- if it equals the number of sets in A, it means we found s_i that fits all of the sets; hence we add it to the result list. The complexity of step 4 is a product of the number of items, the sums and the complexity of KXSSP, and since KXSSP is $O(s\sqrt{n})$ – it is $O(s^2 n^{3/2})$, which is the total complexity of the algorithm. At step 5 we return the list.

3.4 Completeness/ Correctness
3.4.1 Initialization

For $i = 1$, the invariant is respected: in the first iteration, s could be any of the sums of the A_{1j} sets and, trivially, $s_{min} = s_{max} = s$ since it is the sum of the only set checked.

3.4.2 Maintenance

For $i = k$, given $1 \leq k \leq n-1$, without the loss of generality we take A_k as the set currently handled. There are three possible cases for this k^{th} iteration:

a. $KXSSP(A_k, s_k) = true$ *and counter* $= k-1$, meaning s_k is still a possible solution for A.
b. $KXSSP(A_k, s_k) = true$ *and counter* $< k-1$, meaning s_k is not a possible solution for A, hence s_{k+1} will be checked next .
c. $KXSSP(A_k, s_k) = false$, meaning s_k is not a possible solution for A, hence s_{k+1} will be checked next .

Thus the invariant is preserved.

3.4.3 Termination

At the last iteration, Given $i=n$, the three options above are the same for A_n, and respectively s_n, with an emphasis on option a., in which a solution is not just possible, but guaranteed, whilst the two other options may result in a solution previously found or in a null return value . Hence the algorithm gives us the joined sum s, as expected, if such a sum exists.

4 Results- Cryptographic application

4.1 SSP cryptography

Using the classic SSP in cryptography was one of the earliest cryptosystems created, first shown as the Merkle–Hellman knapsack cryptosystem ([12]), having SSP as a special case of the knapsack problem as shown in [13]. This basic cryptosystems was based on creating a super-increasing set (meaning every number is greater than the sum of all of its smaller numbers), and then solving SSP in a greedy polynomial time algorithm. This cryptosystem was broken in [14], and since more complex versions of SSP were used to create stronger cryptosystems as shown in [15]. Here we suggest a use for the MISSP as a relatively strong cryptosystem based on two symmetric private keys.

4.2 MISSP encryption

As mentioned above, the multiple integrated sets SSP (denoted here as MISSP), has a cryptographic application for equal sizes of sets (m), from which we derive the number of sets-n, and an equal number of digits of every integer in the sets (d). In this application, the cipher text is the sequence of the given sets, the private symmetric keys are two of the parameters m, n, and d, and the encrypted plain text is the resulted sum s. For this application, a singular result of s is needed, for an

exact decryption of the cipher text. As seen if Fig.1, the decryption process is constructed of three parts: First, decomposing the cipher text to *n* different sets. In the example of the figure, the cipher text is 55495458205016966826278532461565, K_n is 4, hence the cipher text is decomposed to {55495458, 20501696, 68262785, 32461565}. The second stage is decomposing the resulted sets to *d*-size items. In the example K_d=2, hence the sets are decomposed to the 2D set of 2 digit integers, on which the MISSP algorithm will run, and it is {{55, 49, 54, 58}, {20, 50, 16, 96}, {68, 26, 27, 85}, {32, 46, 15, 65}}. For the last part we run MSSIP on the 2D set.

Its result is the actual plain text. In the example it is 112 for the following summations: 54+58, 96+16, 85+27, and 65+15+32.

Two things are important to mention:

First, the choice of keys of *n* and *d* is, of course, trivially equivalent to any of the combinations of *m* and *d* or *m* and . The decomposition process results the same 2D set, only the decomposition stages are different, and the division of the cipher text (*C*) can be done for the integers first, or for the sets first, this is because $|C| = mnd$ (the size of the cipher text is the product of the item size, the size of every set, and the number of sets).

Second, there is singularity constraint for the resulted plain text, meaning MSSIP necessarily has to have exactly one result. For example, set *A* in table 1 is suitable for MISSP encryption (1 result), but sets *B* (0 results) and *C* (2 results) are not suitable.

More results and examples of using MSSIP encryption are shown in tables 2 and 3.

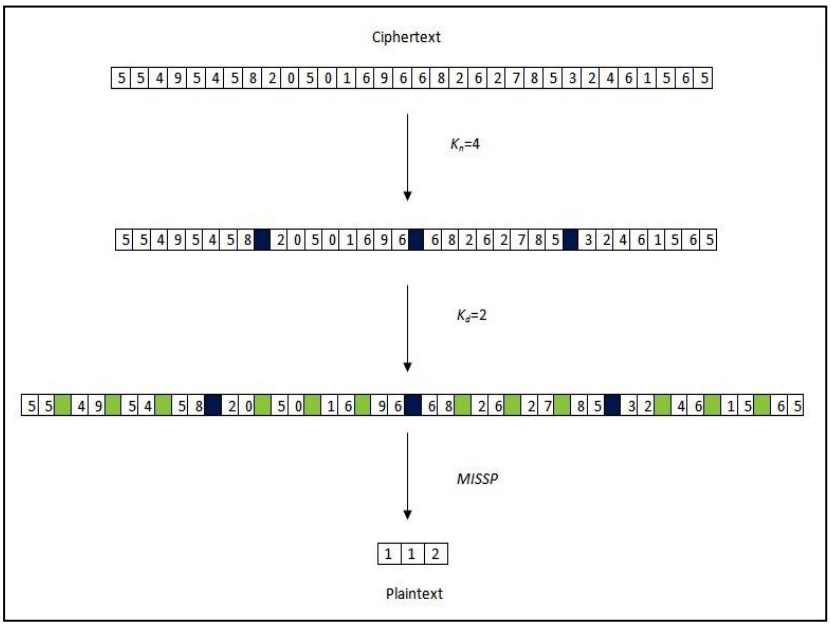

Fig. 1. *MISSP* cipher text decomposed to sets of integers by the private keys, and decrypted by the *MISSP* algorithm

Table 2
MISSP decryption for C=
354931317488131544587365285527407048622982282929545549589854514066
844221964433115138492922182856793451481685316125836929565439313931
659719259012607994665153189731369989753018 95

n/m	*d*	sets	Set items	*s* (plain text)	summations
4/11	4	A_1	{3549, 3131, 7488, 1315, 4458, 7365, 2855, 2740, 7048, 6229, 8228}	26931	3549+7365+2740+7048+6229
4/11	4	A_2	{2929, 5455, 4958, 9854, 5140, 6684, 4221, 9644, 3311, 5138, 4929}	26931	2929+5455+4958+5140+3311+5138
4/11	4	A_3	{2218, 2856, 7934, 5148, 1685, 3161, 2583, 6929, 5654, 3931, 3931}	26931	2218+1685+2583+6929+5654+3931+3931
4/11	4	A_4	{6597, 1925, 9012, 6079, 9466, 5153, 1897, 3136, 9989, 7530, 1895}	26931	6597+9012+1897+7530+1895

Table 3
MISSP decryption for C=
799983342767152577242663441740985671678720845472559646208978678249
295875506162204711109183474250893534771926

n/m	*d*	sets	Set items	*s* (plain text)	summations
4/9	3	A_1	{799, 983, 342, 767, 152, 577, 242, 663, 441}	2942	342+767+152+577+663+441
4/9	3	A_2	{740, 985, 671, 678, 720, 845, 472, 559, 646}	2942	985+678+720+559
4/9	3	A_3	{208, 978, 678, 249, 295, 875, 506, 162, 204}	2942	978+678+249+875+162
4/9	3	A_4	{711, 109, 183, 474, 250, 893, 534, 771, 926}	2942	711+534+771+926

5 Conclusion and future work

In this paper a problem of finding a mutual subset sum for integrated sets (MISSP) was presented, in which we need to find a subset sum that is reachable for all of the sets in a 2D set, if such a sum exists. A relatively efficient algorithm

for solving the problem along with several examples and results were presented. For equal sizes of numbers and sets, a relatively strong cryptosystem with symmetric private keys was presented, along with several decryption examples for it.

Some problems exist in MISSP; the first one is the possibility of getting ambiguous results for the plain text, if more than one possible sum exists for the 2D sets. This problem is addressed by choosing specific injective ciphers, which solves the problem, but creates a bigger work load for the algorithm, in finding those specific ciphers. Another problem is the creation of plain text. It is possible to encrypt in a naïve manner, by finding the first set-items that are conjoined to the needed sum, and than simply adding an alternate offset for adding and reducing from the original items. This cipher is relatively strong, since the SSP is *NP-Complete* and without the private keys, the search is basically futile, specifically if there are several singular results for different *m*, *n*, and *d*'s. The ambiguity of it makes it hard to determine which plain text is the right one, but a stronger encryption is generating random sets, from which we apply our desired plain text (which is *s*). This method is, of course the preferred one, but it is much more complex in its application, since the MISSP has to run consecutively until it finds the desired result. This method was used in achieving the results of this paper, and the plain texts presented in the examples were ones generated by the random sets choices given the keys *m*, *n*, and *d*. Aiming towards a specific plain text in a random generator of the above describe sort, is an action that could either take long computational times, or create a data stack overflow. The mentioned random generator, along with the MISSP algorithm implementation, was written in a Java system for the purpose of this research. Improving it, and the sets randomize generator, along with finding other applications for MISSP and other variations of it, are currently a work in progress.

References

[1] M.R. Garey, D.S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W.H. Freeman, 1979.

[2] T.H. Cormen, C. E. Leiserson, R. L. Rivest, Stein, Clifford (2001) [1990]. "35.5: *The subset-sum problem, Introduction to Algorithms* (2nd ed.), MIT Press and McGraw-Hill.

[3] E. Horowitz, S. Sahni, Computing partitions with applications to the knapsack problem, *Journal of the Association for Computing Machinery*, **21** (1974), 277–292. https://doi.org/10.1145/321812.321823

[4] D. Pisinger, Linear Time Algorithms for Knapsack Problems with Bounded Weights, *Journal of Algorithms*, **33** (1999), no. 1, 1–14.

https://doi.org/10.1006/jagm.1999.1034

[5] K. Koiliaris, C. Xu, A Faster Pseudopolynomial Time Algorithm for Subset Sum, (2015). arXiv*:*1507.02318

[6] D.S. Johnson, Approximation algorithms for combinatorial problems, *Journal of Computer and System Sciences*, **9** (1974), no. 3, 256-278 https://doi.org/10.1016/s0022-0000(74)80044-9

[7] H. Kellerer, U. Pferschy, M.G. Speranza, An effiicient approximation scheme for the subset-sum problem, Chapter in *International Symposium on Algorithms and Computation*, Springer Berlin Heidelberg, Vol. 1350, 1997, 394-403. https://doi.org/10.1007/3-540-63890-3_42

[8] A. Caprara, H. Kellerer, U. Pferschy, The Multiple Subset Sum Problem, *SIAM Journal on Optimization*, **11** (2000), no. 2, 308-319. https://doi.org/10.1137/s1052623498348481

[9] G.J. Woeginger, Y. Zhongliang, On the equal-subset-sum problem, *Information Processing Letters*, **42** (1992), no. 6, 299-302. https://doi.org/10.1016/0020-0190(92)90226-l

[10] M. Cieliebak, S. Eidenbenz, A. Pagourtzis, K. Schlude, On the Complexity of Variations of Equal Sum Subsets, *Nord. J. Comput*., **14** (2008), no. 3, 151-172.

[11] C. Bazgan, M. Santha, Z. Tuza, Efficient approximation algorithms for the Subset-Sums Equality problem, Chapter in *International Colloquium on Automata, Languages, and* Programming, Springer Berlin Heidelberg, 387-396, 1998. https://doi.org/10.1007/bfb0055069

[12] R. Merkle, M. Hellman, Hiding information and signatures in trapdoor knapsacks, *IEEE Transactions on Information Theory*, **24** (1978), no. 5, 525–530. https://doi.org/10.1109/tit.1978.1055927

[13] S. Martello, P. Toth, 4 Subset-sum problem, *Knapsack Problems: Algorithms and Computer Interpretations*, *Wiley-Interscience*, 1990, 105–136.

[14] A. Shamir, A polynomial-time algorithm for breaking the basic Merkle - Hellman cryptosystem, *IEEE Transactions on Information Theory*, **30** (1984), no. 5, 699–704. https://doi.org/10.1109/tit.1984.1056964

[15] A. Kate, I. Goldberg, Generalizing cryptosystems based on the subset sum problem, *International Journal of Information Security*, **10** (2011), no. 3, 189-199. https://doi.org/10.1007/s10207-011-0129-2

Received: September 19, 2017; Published: October 19, 2017