Pyramix: A Multi-Purpose Trapdoor for Knapsacks

abstract: Pyramix is a new trapdoor function for knapsack-type cryptosystems, enabling public key encryption, public key watermarking, and two-way secret key encryption.Knapsacks with a large range for the plaintext-numbers are used to achieve optimal density ~1. In public key encryption and watermarking, a mixing matrix protects the public key numbers against attacks based on specific properties of the primary trapdoor mechanism. In two-way secret key mode, the dimension of the knapsack can be low (~ 20), enabling high speed key set-up, encryption and decryption.

key words: public key, knapsack problem, knapsack density, trapdoor function, two-way secret key, watermark.

Various proposals have been made to turn the knapsack problem into a public key system, but these were either broken or required impractically large parameters. Yet, because knapsack encryption can be done much faster than, for instance, RSA-encryption, a viable knapsack sytem would be a valuable addition to existing systems.
A 'knapsack', in its simplest form, is the sum of a secret subset of integers chosen from a larger, public set of integers. The subset represents the plaintext message, the sum is the encrypted message.
When a knapsack system caves in, the blame inevitably falls on the trapdoor mechanism connecting the public key with the secret key. Either a backdoor is found through which the secret key can be deduced from the public key, or the trapdoor requires public key numbers that are so large that the 'density' of the knapsack is low. Low-density knapsacks can be decomposed into the original integers in managable computing time.
Pyramix generates knapsacks approaching the optimal density 1, with public key numbers that are only moderately restricted by boundary conditions. Slightly modified, Pyramix generates knapsacks with multiple solutions which can be used as individual watermarks, verifiable with a public key.
Pyramix also enables two-way secret key encryption, in which there is no public key and both parties involved in a communication stay ignorant of each other's keys.
While the dimension of the knapsack (the number of integers summed) in the public key mode has to be ~ 100 and up, in two way secret key mode, the dimension can be as low as 20, resulting in very fast key set-up, encryption and decryption. So even 'throw-away key mode', in which all keys are discarded after one time use, becomes an attractive option.

Ia. The Trapdoor's Core

nb: unless stated otherwise, all variables and constants are non-negative integers. Numbers (scalars) are represented by lower case letters, vectors and matrices by upper case letters

The basic knapsack problem requires finding the unique subset $\{p_{s1}, p_{s2}, \ldots p_{sm}\}$ from the complete – and publicly known - set $\{p_1, p_2, \ldots, p_n\}$ for a given $y = p_{s1} + p_{s2} + \ldots + p_{sm}$
A more general knapsack-type expression is

$$y = \sum_{j=1}^{n} p_j x_j \quad . \tag{1.1}$$

in which the x, representing the plaintext message, are confined to a range 0….r.
Such a knapsack can be interpreted as the inproduct of an n-dimensional public key vector P and a plaintext vector X:

$$y = P^T * X \quad . \quad \text{('T' denotes the transposed vector or matrix)} \tag{1.1a}$$

The core of the Pyramix trapdoor mechanism is a 'pyramid'of lineair equations. Each equation has the form:

$$y = ay_1 + by_2 \quad \text{with a and b relative prime (gcd(a,b)=1).} \tag{1.2}$$

This equation, for a given y, has a unique solution (y_1, y_2) in the range (0….b-1, 0…..a-1).
The y_1 and y_2 can themselves be the result of a similar equation, yielding, for a given y, two unique pairs of solutions (y_3,y_4) and (y_5,y_6) within ranges (0…b'-1, 0….a'-1) and (0…b"-1, 0….a"-1).
For a unique solution:

$$a'y_3 + b'y_4 < b - 1 \quad \text{and similar for a, a", b" .} \tag{1.3}$$

This nesting mechanism can be continued, resulting in a 'pyramid' of lineair equations with 2^n terms in the n_{th} layer. The minimum size of the a's and b's, because of the uniqueness condition (1.3), increases exponentially from bottom to top.
The three-layer pyramid below illustrates this mechanism:

$$y = 141{,}201y_1 + 141{,}202y_2 \tag{1.4}$$
$$y_1 = 270y_3 + 271y_4 \qquad y_2 = 269y_5 + 272y_6$$
$$y_3 = 10x_1 + 19x_2 \quad y_4 = 11x_3 + 18x_4 \quad y_5 = 12x_5 + 17x_6 \quad y_6 = 13x_7 + 16x_8$$

The variables in the bottom layer have been labeled x_i , because they will become the input-slots for the plaintext message X. Such a pyramid can be looked at from two sides: from the bottom, it is a knapsack-expression

$$y = \sum_{i=1}^{8} s_i x_i \ . \tag{1.5}$$

with $s_1 = 141{,}201 \times 270 \times 10 = 381{,}242{,}700 , \ldots, \ s_8 = 141{,}202 \times 272 \times 16 = 614{,}511{,}104$ (1.5a)

or, $y = S^T * X$. (1.5b)

Viewed from the top, the pyramid is a recursive expression of the form (1.2): If one knows a and b in such an equation, it can be solved efficiently as follows: first, an auxillary constant h is calculated from:

$$ah = 1 (\mathrm{mod}\, b) \ . \tag{1.6}$$

This can be done quickly, even for very large numbers, using Euclid's algorithm.
h can be seen as the inverse of a(mod b), and is in effect a derived secret key number. If h is known:

$$ay_1 = y (\mathrm{mod}\, b) \Rightarrow y_1 = yh (\mathrm{mod}\, b) \Rightarrow y_2 = (y - ay_1) \div b \ . \tag{1.7}$$

Once y_1 and y_2 are known (and the a's and b's in the next layer) the same procedure can be used to find y_3, y_4 and y_5, y_6. Whatever the number of layers, this process can be continued all the way down till one finds the x_i, i.e. the message vector X.
This is a trapdoor in which a message x_i is encrypted with the numbers s_i, while for decryption one uses the a's and b's in every layer. However, this 'bare' trapdoor can not be used in public key mode.
In public key mode, the s_i would serve as public key numbers, and the a's and b's in every layer as the secret key numbers. Security hinges on the impossibility to derive the secret key numbers from the public ones. A look at the structure of the s_i , as illustrated by $s_1, \ldots s_8$ in equation (1.5a), shows that this is not the case, because there is a wealth of subsets of s_i that have one or more secret key numbers as a common divisor. Tests have shown that simply tabulating the greatest common divisor (gcd) of all pairs (s_i, s_j) usually betrays the larger secret key numbers (there will be false positives, but not nearly enough to provide security against this kind of attack). Therefore, a second piece of architecture is needed, the mixing matrix.

Ib. The Mixing Matrix

The mixing matrix creates public key numbers that are secure against the gcd-attack. Moreover, given a properly composed matrix, we have not been able to extract any other information from the public key that could provide a shortcut to the 'pyramid' numbers. The (n-dimensional) public key vector P is constructed according to :

$$P = A^T * S \ . \tag{1.8}$$

in which a secret nxn matrix A and its transpose A^T are introduced. A has to be invertable. We further restrict A to unimodular matrices (detA=1) with elements that are either 0 or 1. (see appendix A) Though this is not strictly necessary, it is convenient because the inverse of A will then also have only integer elements (possibly negative). The function of A^T is to create public key numbers p that are each the sum of a small subset of secret key numbers s. If A^T is set up properly, no p consists of s-numbers derived exclusively from either the left or the

right side of the pyramid. Such an A^T ensures that no subset of p's has a large common divisor that could betray one of the secret key numbers in the pyramid.
Reconstructing S from P would mean finding matrix A. The number of possibilities for A in an exhaustive search is ~ (n over $\lambda)^n$, with λ the number of non-zero elements in each row (λ must be set in advance, because it changes the range for the input numbers of the pyramid from r to λr). Exhaustive search is out of the question even for low dimension n, but smarter efforts are certainly possible, and care must be taken to avoid 'weak' mixing matrices. (a more in depth treatment of this subject is in preparation).
(For decryption, see section II)

Ic. Knapsack Size and Density

The uniqueness condition (1.3) forces the pyramid secret key numbers up exponentially from one layer to the next. For a pyramid with l layers (so the dimension of the knapsack $n = 2^l$) and a range r for the message numbers, the size is approximately:

$$2^{n-l-1}(r+1)^{n-1} \quad . \tag{1.9}$$

An important parameter characterising the resistence of a knapsack to 'head on'-attacks (directly solving the primary equation $z = P * X$, for instance with the LLL-algorithm) is the density, defined as:

$$d = \frac{n\log(r+1)}{\log(\max p)} \quad \text{(max p = largest public key number; d is not an integer).} \tag{1.10}$$

knapsacks with d>1 generally don't have a unique solution and can't be attacked in this way; vulnerability to 'head on'-attacks increases quickly with d sinking below 1.
Combining (1.9) and (1.10), the density of pyramid knapsacks is:

$$d = \frac{n\log(r+1)}{(n-l-1)+(n-1)\log(r+1)} \quad \text{(all logarithms base 2)} . \tag{1.11}$$

Adding a mixing matrix with parameter λ changes the range for the input numbers to λr, and increases max p (1.9) to:

$$\lambda 2^{n-l-1}(\lambda r+1)^{n-1} \quad . \tag{1.12}$$

$$\Rightarrow d = \frac{n\log(r+1)}{\log\lambda+(n-l-1)+(n-1)\log(\lambda r+1)} \quad . \tag{1.13}$$

For large r, the r-terms dominate. If n is also large:

$$d \approx \frac{\log r}{\log r+\log\lambda} \quad . \tag{1.14}$$

The size of the encrypted message y will be approximately:

$$nr\max p \quad \text{, dominated by the size of max p.} \tag{1.15}$$

II. Public Key Encryption and Decryption

The sender, Alice, only knows the public vector P. She 'knapsacks' her message X in the usual way:

$$y = P^T * X \quad . \tag{2.1}$$

Using (1.8), this equation can be rewritten:

$$y = P^T * X = S^T * A * X = S^T * K \quad \text{(using } P = A^T * S \Leftrightarrow P^T = S^T * A\text{)} \tag{2.2}$$

so, $y = S^T * K$. (2.3)

with $K = A * X$ (2.4)

The legitimate receiver Bob will decrypt y in the usual way, ((1.2),(1.6)(1.7)) but.as is evident from equation (2.3), he will not find the message vector X, but vector K. Decryption is only completed when Bob calculates:

$$X = A^{-1} * K \quad . \tag{2.5}$$

The x_i are numbers from 0 to range r. The k_i need to have a definite range as well (to allow proper decryption), which restricts the number of non-zero matrixelements (1's) in each row of A. The maximum number of such elements is a parameter λ to be chosen. In our implementation, we chose λ=3, the minimum value to enable safe encryption, and, moreover, to have a fixed number of three 1's in every row.

Table I

Density d and magnitude of the encrypted message [y] (expressed as the number of decimal digits) for a range of n and r, with λ = 3. For larger λ, the density decreases slightly. [x] is the size of the plaintext.

n =	**16**	**32**	**64**	**128**
r=				
1	d = 0.53	0.51	0.50	0.50
	[y] = 15	29	58	116
	[x] = 5	10	19	38
9	0.74	0.71	0.70	0.70
	29	57	112	225
	16	32	64	128
99	0.86	0.83	0.82	0.81
	45	89	178	356
	32	64	128	256
999	0.92	0.89	0.88	0.87
	61	122	242	485
	48	96	192	384
9999	0.95	0.92	0.91	0.90
	77	154	306	612
	64	128	256	512
99999	0.97	0.94	0.93	0.92
	93	186	370	741
	90	160	320	640

So, for high density, a large range is needed. Instead of letting the x_i represent single bits (r=1), they could be bytes or blocks of decimal digits. The Pyramix block size for practical n is then fairly large, ~ 1000 bits.

Algorithms like LLL attack the primary knapsack-equation $z = P * X$, disregarding the nature of the trapdoor and the public key. We tested our implementation of Pyramix (with λ=3) against the LLL-algorithm as published by Lenstra et al., on a standard, ~ 1 Ghz laptop computer. The results are listed below:

Table II
For every n and r, we compared one or more Pyramix-knapsacks with knapsacks of equal density but a random vector P.

# (number of tests)	n	r	t (min)	cracked	vector size (min/max)	d	Pyramix/random
1a (5)	8	999	.21+/-.03	+	~ $10^3/10^4$	.92	Pyramix
1b (5)	8	999	.21+/-.03	+	~$10^3/10^4$	.92	random
2a (5)	16	999	1.9+/-.15	+	~$10^3/10^4$	.86	Pyramix
2b (5)	16	999	1.8+/-.06	+	~$10^3/10^4$	.86	random
3a (4)	32	999	22+/-1	+	~$10^3/10^4$	.83	Pyramix
3b (3)	32	999	23+/-1	+	~$10^3/10^4$	.83	random
4a (1)	32	99999	44	+	~$10^5/10^6$	.90	Pyramix
4b (1)	32	99999	49	+	~$10^5/10^6$	.90	random
5a (1)	64	999	450	+	--	.81	Pyramix
5b (1)	64	999	323	+	--	.81	random

We conclude, that for an LLL-attack, the Pyraset public key numbers are equivalent to (pseudo)random numbers.

III. Two Way Secret Key Encryption

Pyramix enables encryption with neither a public key or a shared secret key. The message is sent three times over the insecure channel: first, from Alice to Bob encrypted with Alice's secret key, back to Alice doubly encrypted with both Alice's and Bob's secret key, and finally, partially decrypted by Alice, back to Bob.

First, Alice breaks up her message into m blocks, each consisting of n integers in the range $0 \ldots r_a$. So a unit message is mxn integers. Each block is 'knapsacked':

$$y_k = \sum_{i=1}^{n} x_{i,k} s_i \qquad \text{k=1….m .} \tag{3.1}$$

or $Y_{(s)} = X * S$ (X is an nxm-matrix). (3.1a)

S is Alice's secret key vector. Bob receives vector $Y_{(s)}$, and subjects this to his own knapsack operation:

$$z = T^T * Y_{(S)} \quad . \tag{3.2}$$

T is Bob's secret key vector, analogous to Alice's S. Bob sends z to Alice. She can switch summations:

$$z = T^T * Y_{(S)} = T^T * (X * S) = (T^T * X) * S \tag{3.3}$$

$$\Rightarrow z = Y^T{}_{(T)} * S \quad . \tag{3.4}$$

with $Y_{(T)} = X^T * T$. (3.5)

Alice has rewritten z as a knapsack of her own secret key vector S with a new, n-dimensional vector $Y_{(T)}$.

She can efficiently extract this $Y_{(T)}$ from the knapsack, using her secret pyramid key numbers. She then sends $Y_{(T)}$ to Bob. Each component is a knapsack of plaintext message-numbers and Bob's own secret key numbers:

$$y_{(T)i} = \sum_{k=1}^{m} x_{i,k} t_k \quad . \tag{3.6}$$

Bob has no trouble decrypting the $y_{(T)\,i}$, using his secret pyramid numbers.
As no public key is involved, there is no need for a mixing matrix, and the dimension of the knapsacks can be chosen substantially lower, ~10-30. Alice and Bob have to share some information, but this can be done publicly. They must agree on values for n and m and ranges r_x and r_y.
Construction of the vectors S and T starts with setting the range r_x for Alice's message numbers x_{ij}. Bob, knowing r_x and m, sets up his pyramid with $l_m = {}^2\log m$ layers, generating T. The size of his secret key numbers t (the components of T), as a consequence of the Pyramix set-up, will be approximately:

$$[t] = 2^{m-l_m-1}(r_x+1)^{(m-1)} \qquad \text{(for m=16, } r_x = 9,\ [t]\sim 2*10^{18}\text{).} \tag{3.7}$$

and the size of the $y_{(T)}$'s which Alice has to extract from knapsack z:

$$[y_{(T)}] = m r_x [t] \qquad \text{(for m=16, } r_x = 9,\ [y]\sim 3*10^{20}\text{) .} \tag{3.8}$$

Alice has to set up her pyramid with a range $r_y > [y_{(T)}]$ to ensure proper decryption, so:

$$r_y > m r_x 2^{m-l_m-1}(r_x+1)^{m-1} . \tag{3.9}$$

Because Alice does not know Bob's key, an upper bound r_y must be established which leaves enough headroom for Bob to set up his pyramid generating T with all components smaller than r_y. With m=16, r_x = 9, an extra factor of 10 would probably suffice, so $r_y \sim 10^{21}$.
Alice then chooses some n (smaller than m – see below) to set up her own pyramid, generating S. Despite the fact that n is small, the components of S, and therefore Y, will be relatively large, because the pyramid is set up to accept input-numbers with a large range r_y:

$$[y_{(S)}] = n r_x [s] \approx 2^{n-l_n-1} n m^{n-1} r_x^n \; 2^{(n-1)(m-l_m-1)} (r_x+1)^{(m-1)(n-1)} . \tag{3.10}$$

(using (1.7) with $r=r_y$)

As a public key system, knapsacks with dimension ~20 would be much too small to provide security. In two way secret key mode, only the knapsacks themselves get into the insecure domain were Eve is listening in: first as the m-dimensional vector $Y_{(S)}$, then as the number z, and finally as the n-dimensional vector $Y_{(T)}$. These are n+m+1 quantities possibly known to Eve, while the number of unknowns is n+m (secret key numbers) +nxm (message numbers).

One apparent weakness deserves attention here: when Bob sends his doubly encrypted message $z = T * Y_{(S)}$ back to Alice, Eve knows both z and, from the first encrypted message, $Y_{(S)}$. This is a knapsack in which only vector T is unknown, so it looks as if it can be attacked with the LLL-algorithm, and very efficiently because of the advocated low dimension m. However, substituting the relevant parameters in (1.8), using (3.7) and 3.10), the density of z is:

$$d = \frac{m \log([t]+1)}{\log(\max[y_{(S)}])} \approx \frac{m[q]}{n[q] + (n-1)\log m + (n-l_n-1) + \log n + n\log r_x - [q]}$$

$$\text{with } [q] = (m-l_m-1) + (m-1)\log(r_x+1) . \tag{3.11}$$

for r_x, m and n in the range 10-30, in fairly good approximation:

$$d \approx \frac{m}{n} \quad . \tag{3.12}$$

This explains why Alice must choose n smaller than m: this raises the density of knapsack z above 1, making it resistant to LLL-type attacks. Exactly how high d should rise to secure z, as a function of n and m, needs to be studied in more detail.

IV. Public Key Watermarking

Pyramix can generate large collections of individual 'public key watermarks'. Each watermark is a set of numbers for which anyone can check that they comply with a certain rule, while it is practically impossible to create another, valid set without knowledge of the secret key.
In Pyramix public key encryprion, the trapdoor provides secret key numbers that ensure a unique solution vector X to the knapsack-problem y=P*X. Modification of these secret key numbers and the boundary conditions results in a knapsack and trapdoor that generates numerous vectors X_i that all produce the same y with a single public key P:

$$y = P * X\_1 = P * X\_2 = ...P * X\_l \quad . \tag{4.1}$$

If y and P are public, for a given X_i anyone can check that this vector belongs to the collection of solutions, simply by computing P*X_i = y . But without the secret key, it is still unfeasible to solve this equation to find other X_i in the collection. For encryption, the basic equation (1.2),

$$y = ay_1 + by_2$$

has boundary conditions to ensure a unique solution in a certain range. Allowing a larger range for y_1 and y_2 gives multiple solutions for every possible y. To avoid negative integers, we use a slightly different equation for a watermarking pyramid:

$$y = ay_1 - by_2 \quad . \tag{4.2}$$

For instance, $167 = 12y_1 - 13y_2$ has solutions (41,25) , (54, 37), (67,49) etc. In general, if (5.2) has a solution ($\underline{y}_1$,$\underline{y}_2$) for a given $\underline{y}$, then the pairs ($\underline{y}_1 + n*b$, $\underline{y}_2 + n*a$) are also solutions for this $\underline{y}$.
Like in encryption, one can build a pyramid of equations on this principle. Consider this 3-layer pyramid:

$$y = 45\,y_1 - 44\,y_2 \tag{4.3}$$

$$y_1 = 20\,y_3 - 19\,y_4 \qquad y_2 = 22\,y_5 - 21\,y_6$$

$$y_3 = 10k_1 - 9k_2 \qquad y_4 = 8k_3 - 7k_4 \qquad y_5 = 13k_5 - 12k_6 \qquad y_6 = 15k_7 - 14k_8$$

(the secret key numbers in each pair are consecutive, so that the equations have a solution for *every* given y. As all the alternatives for the y's in one layer must be possible y's for the layer below, this is a useful property for constructing such a pyramid. The variables in the lowest layer have been named k for a reason explained below)

The components of the secret key vector S in this example are : $s_1 = 9000$ (= 45x20x10) , $s_2 = 8100$, $s_3 = 6840$, $s_4 = 5985$, $s_5 = 12,584$, $s_6 = 11,616$, $s_7 = 13,860$, $s_8 = 12,936$
Choosing the range for the k as 1…..20,
any combination of the following pairs satisfy y = 281:
(k_1,k_2) = (17,17) , (8,7), (9,6), (19,15), (10,5), (11,4), (14,5) *(7 alternatives)*
(k_3, k_4) = (17,17) , (10,9), (3,1), (16,13), (9,5), (14,9), (7,2), (14,4),(20,9),(13,1) *(10 alts)*
(k_5, k_6) = (11,11) , (20,19), (8,6), (17,14), (5,1), (14,9), (11,4) *(7 alts)*
(k_7, k_8) = (11,11) , (19,18), (5,3), (13,10), (7,2), (15,9) *(6 alts)*

So, for any K_j assembled from the list above:

$$S * K\_j = y = 281 \quad . \tag{4.4}$$

Of course, at this point one does *not* have 7x10x7x6 = 2940 different vectors K that can serve as watermarks, because each pair varies independently of all the others and shows at most 10 variations. Someone who is aware

of this and has seen no more than two completely different K, say {17,17, 10,9, 8,6, 13,10} and {8,7, 3,1, 17,14, 7,2} can construct 2^4-1 other K, for instance {17,17, 3,1, 8,6, 7,2}, that satisfy y=281 as well. However, the mixing matrix A, which is necessary anyway to protect the composite secret key numbers S, takes care of this as well. Like in (1.8), the public key vector will be:

$$P = A^T * S \quad . \tag{4.5}$$

and the public watermark vectors, following (2.5):

$$X\_j = A^{-1} * K\_j \tag{4.6}$$

For knapsacks of dimension n, the number of message vectors producing the same y can be easily $\sim 10^{(n/2)}$, with the total number of possible messages $\sim 10^n$. So for n~100, there are a lot of watermark vectors available, while their fraction of the total number is very low. Using a large percentage of available watermark vectors would probably compromise security, but in practice this will never be a problem. Perhaps extra rules, selecting only K-vectors 'far apart' should be introduced, but we leave all this for later work.

Appendix A: Unimodular 0/1-Matrices

It is convenient to have an efficient construction method for nxn unimodular matrices with elements either 0 or 1. (A1) is true for matrices of any even dimension (suppressed elements are zero):

$$\begin{Vmatrix} a & b & & & & \\ c & d & & & & \\ & & a' & b' & & \\ & & c' & d' & & \\ & & & & a'' & b'' \\ & & & & c'' & d'' \end{Vmatrix} = \begin{Vmatrix} a & b \\ c & d \end{Vmatrix} \times \begin{Vmatrix} a' & b' \\ c' & d' \end{Vmatrix} \times \begin{Vmatrix} a'' & b'' \\ c'' & d'' \end{Vmatrix} \quad . \tag{A1}$$

There are six 2x2 matrices with 0/1-elements and determinant +/- 1:

$$\begin{Vmatrix} 1 & 0 \\ 0 & 1 \end{Vmatrix} = \begin{Vmatrix} 1 & 0 \\ 1 & 1 \end{Vmatrix} = \begin{Vmatrix} 1 & 1 \\ 0 & 1 \end{Vmatrix} = 1 \quad \text{and} \quad \begin{Vmatrix} 0 & 1 \\ 1 & 0 \end{Vmatrix} = \begin{Vmatrix} 0 & 1 \\ 1 & 1 \end{Vmatrix} = \begin{Vmatrix} 1 & 1 \\ 1 & 0 \end{Vmatrix} = -1 \ . \tag{A2}$$

These building blocks can be used to construct even-dimensional nxn matrices of the form (A1) with determinant 1. This allows $1/2x6^{n/2}$ diiferent 'backbones' for such a matrix.
To create mixing matrices that comply with the boundary conditions mentioned in section Ib, one uses the general property that a determinant is not changed by internally adding rows or columns:

$$\begin{Vmatrix} a & b & c \\ d & e & f \\ g & h & i \end{Vmatrix} = \begin{Vmatrix} a+b & b & c \\ d+e & e & f \\ g+h & h & i \end{Vmatrix} \ . \tag{A3}$$

Omitting further details, it should be obvious that this allows one to construct sufficient numbers of nxn unimodular 0/1 matrices with a fixed number of 1's per row, per column or both, with additional conditions for the distribution of the 1's over certain sections of the matrix, etc.