GLASS : STOCHASTIC SUBSET CRYPTO PRIMITIVE

This system uses noise to "shatter" or "smear" the information in the plaintext over a ciphertext p that is therefore n times larger than the plaintext p. Each symbol in c contains $\frac{1}{n}$ bits of information about p.

This is achieved by randomly dividing both P and C into sets T_i and S_i represpectively. In each case $|T_i| = \frac{1}{2}|P|$ and $|S_i| = \frac{1}{2}|C|$. Let $X_i = T_i$ or T_i^c, depending on whether $c_i \in S_i$ or $y_i \in S_i^c$, where c_i is the ith component of the ciphertext. Then our construction of the T_i and the S_i guarantees that $\cap X_i = \{p\}$.

To encode some $p \in P$, one checks for each T_i whether $p \in T_i$. If so, then let $c_i = f_i(p)$ be a random element in S_i. If $p \notin T_i$, then let $c_i = f_i(x)$ be some random element of S_i^c. So each c_i encodes 1 bit of information about p in the form of whether or not that row is or is not in S_i. Since c_i is n bits long, and all of the bits matter, we have each symbol in c_i worth $\frac{1}{n}$th of a bit.

We can decode $c = f(p) = (c_0, ..., c_{n-1})$ by forming the binary vector $\chi_S(c)$ that encodes whether $c_i \in S_i$ for each i. If $\chi_S(c) = (0,0,1,1,0,1,0,1)$, then one need only look for p such that $\chi_T(p) = (0,0,1,1,0,1,0,1) = \chi_S(c)$.

For each bit of plaintext, we need a different random division of P into sets T_i and T_i^c, as well as random division of C into sets S_i and S_i^c. To create f, we need to create each f_i such that $f_i(x) \in S_i \iff x \in T_i$. Then $f(x) = f_0(x) \frown ... \frown f_{n-1}(x)$ or (in vector language) $(f_0(p), ..f_{n-1}(p)) = c$.

There's an efficient way to do this. Generate the T_i by creating a permutation $\Psi : P \to P$. If you represent this as a bit matrix M, where row i contains the binary representation of $\Psi(i)$, then the columns of M can be read as characteristic (indicator) functions of the desired T_i sets. A similar trick using a function ζ works to get the S_i sets. One can easily use Ψ and ζ to encode and decode, and no further compression seems to be possible.

Most recently I used $n = 16$. So Ψ and ζ both require $65536 * 16 = 1048576$ bits each. That's a heavy key. Smaller versions of the system are much cheaper. There are $((2^{16})!)^2$ keys for a 16 bit system.

The first C version presented on YouTube also includes a random permutation of bits for each row as well as a mixing of the c_i. This adds very little to the weight of the key, and should further obscure which bits are associated with the unknown S_i.