The Ultimate Transposition Cipher (UTC)

Complementing Erosive Intractability with a Durable Entropic Advantage

Gideon Samid
Department of Electrical Engineering and Computer Science
Case Western Reserve University
Gideon.Samid@case.edu

Abstract: The Ultimate Transposition Cipher (UTC) is defined as an encryption algorithm E, and its reverse, E^{-1}, such that for X and Y, two arbitrary permutations of a list of an arbitrary size n elements, there is a key, k such that: $Y=E_k(X)$, and $X=E^{-1}_k(Y)$. One would protect a plaintext P by concatenating it with false messages, "decoys", D = $d_1, d_2,...d_{m-1}$: to construct a Pre-Transposition Plaintext (PTP): P, $d_1, d_2,...d_{m-1}$. Then one would "mix" the result via a UTC, to create a ciphertext C: C = UTC(P, $d_1, d_2,...d_{m-1}$). The intended recipient, aware of the value of the key, k, used to generate C, would decrypt C to P, $d_1, d_2,...d_{m-1}$, and dismiss the decoys. A cryptanalyst will identify at least m keys, one decrypting C to P, $d_1, d_2,...d_{m-1}$, and at least (m-1) others, decrypting C to various permutations, like: $d_{i,}$ d_j ,d_l , P, d_t ...., and will go no further, because C projects mathematical parity towards these m plausible candidates for the actual message sent. We show that in real life situations when both sides can reasonably prepare a list of plausible plaintexts, the UTC equals the mathematical security offered by Vernam's One-Time-Pad, albeit, without Vernam's key size inconvenience. UTC decoys may be constructed manually or with AI. Applying a UTC protection before further encrypting with any common cipher will add a new dimension of equivocation (a clear entropic advantage) to the prevailing intractability-only protection. An example for an actual UTC is referenced and described.

1.0 Introduction

Modern cryptography hinges on assumed -- unproven -- intractability of its selected ciphers. And we rarely switch them around either. A well-funded long-term, sophisticated, maybe desperate adversary would regard such popular intractability-based ciphers as a 'survival security' priority, applying enormous resources towards a mathematical or a computational compromise. What's more, as history has shown, the greater the effort to crack a cipher, the greater the effort to hide the fact that the cipher has been broken. This little mentioned fact munches on the confidence of chief security officers, and recent allegations regarding the NSA have not helped any.

In other words, modern cryptography is based on erosive intractability where the rate of erosion is anybody's guess.

This situation leads one to search for more reliable methods to satisfy the aims of cryptography. Come to think about it, all commonly generated ciphertexts do commit to the singular plaintext that generated them. Say then, that a persistent adversary capturing a ciphertext will eventually extract the underlying plaintext. We hope that our adversary will use nothing more efficient than brute force. Alas, a variety of accelerated brute force methods may be devised and quietly surprise us. The very idea that modern cryptography hinges on the assumption that our adversary will not innovate a mathematical/computational shortcut is very disturbing -- betting against innovation, in this day and age!

This inherent vulnerability of erosive cryptographic intractability is a strong motivation to think afresh and come up with a different principle to establish cryptographic defense.

This thought brings to the fore the old Vernam (One Time Pad) cipher: it offers mathematical security via its durable entropic advantage -- equivocation.

Cryptographic equivocation will be defined as a state where an all powerful cryptanalyst possessing a certain ciphertext, will not be able to determine without a reasonable doubt the identity of the plaintext that was concealed in this ciphertext.

This applies to Vernam's cipher -- One-Time-Pad -- where a ciphertext comprised of n bits could have been generated by any plaintext, which is n bits long or less. This implies that the only useful information yielded by a properly constructed Vernam cipher is the largest possible size of the plaintext it conceals. Otherwise the cryptanalyst has no more information than she had before capturing the ciphertext.

Vernam offers 'total equivocation' but it comes with a steep price modern cryptography is generally not prepared to pay: the tedium of the key, which must be as large the plaintext, it conceals.

Let’s focus on the ground between these two extremes: zero equivocation (ciphers based on erosive intractability) and full equivocation (One-Time-Pad). Over this equivocation increases from zero to "100%," so to speak. What is left for us is to find a cipher that would home in on this fertile ground, and do so with less practical punishment (more convenient key) than Vernam.

This article defines a class of such ciphers, and it references a specific cipher within this class that accomplishes the goal of variable equivocation.

2.0 The Cryptographic Battleground

"*Gentlemen don't read other people's mail*" declared Henry Stimson, Secretary of War (closing down a secret intelligence service before WW-II). Accordingly, gentlemen have no use for cryptography. Or say, one uses cryptography in an environment of suspicion, competition, battle. And hence we should ask ourselves: what is the battleground where we practice this craft.

Let's consider two adversaries competing over a given situation, which we shall call 'the battleground'. The adversaries wish to establish an advantage in the battleground, and hence they wish to gain information about their opponent's design, as well as prevent the opponent from discerning their design. This translates to each party using cryptography to communicate within its parts, and using cryptanalysis to gain intelligence from their opponent's communication within its parts.

The battleground where the adversarial parties wish to establish an advantage is known to both parties, at least to some degree. This is a critical point, cryptographic adversaries don't compete in a vacuum, they compete over a shared battleground.

Suppose now that one's adversary has sent an encrypted message within its various parts. The corresponding ciphertext is captured. The captor may decide to prepare a list of plausible plaintexts corresponding to the captured ciphertext: "what would be a plausible plaintext encrypted into the captured ciphertext?".

Anyone aware of the battleground may prepare such a list of plausible plaintexts. Let there be n such plausible messages in association with a given ciphertext. For illustration purposes let's further assume that each of the n distinct plaintext candidates has the same chance (p_i) of being the actual plaintext encrypted into the captured ciphertext: $p_i = 1/n$ for i=1,2,...n. The entropy of the situation is:

$$H_n = -\Sigma\, p_i \log p_i = \log(n)$$

In the event that n=1, the entropy is zero. To wit: if the situation is so clear, so obvious that the adversary has only one reasonable move, then there is no doubt that the encrypted message expresses this move. There is very little need to actually crack the ciphertext.

On the other extreme n --> ∞, there exists a daunting count of plausible messages, and the entropy is unbound too.

Now suppose the adversary wishes to send message d_j (j=1,2,...n) as the message to be protected by the cipher: d_j =M, the protected message. Further suppose that the adversary concatenates (m-1) options from the remaining (n-1) candidates: and then "mixes" (transposes) the m messages to some given order, and that order is designated as the ciphertext and released in the battleground. We shall designate the pre-transposed sequence of messages, as the Pre-Transposed Plaintext, or PTP:

$$PTP = d_j, d_1, d_2,\ldots.d_{j-1}, d_{j+1},\ldots d_m$$

Where d_i is plausible message i. The intended reader, who knows how to properly "unmix" the ciphertext, will extract the original order, regard the first message d_j as the protected message, M, for him to account for, and disregard the other (m-1) messages. By contrast, the cryptanalyst, applying even unlimited computing power, unlimited time, and wisdom, will, by the end of the day, identify all the mixed m messages, but will have no way to identify d_j as the concealed message M. The ciphertext could be "unmixed" to put any one of the m messages as the one that counts. Nothing in the ciphertext distinguishes d_j as the protected message M.

For the latter statement to be true, it is necessary that the encryption algorithm will be complete. Namely that any two permutations could be designated one as a pre-transposed version and one as a post-transposed version and these two arbitrary permutations will be transformed into each other using a key applied to the transposition algorithm or to its reverse. We call such a complete, or ultimate transposition algorithm: Ultimate Transposition Cipher, or UTC.

If a party transposed m plaintext candidates into a ciphertext using a UTC, then a cryptanalyst will end up, at best, identifying all the m candidates as plausible plaintexts, and will be unable to narrow down this determination.

The entropy of the situation will have come down from $H_n = \log(n)$ to $H_m = \log(m)$. Say then that by capturing the ciphertext, and analyzing it exhaustively, the cryptanalyst has gained an entropic advantage of:

$$H_n - H_m = \log(n) - \log(m) = \log(n/m)$$

On the flip side the party that employs the UTC is submitting to an extra burden of encrypting not one plaintext candidate but m. The larger the value of m, the more effort of carrying out this strategy, but also, the greater the entropic advantage to the user. So one would balance out: more work for more security, or less work for less security.

There are circumstances where the value of n is very high ($n \to \infty$), namely, it is wide open what could have been the message that was actually encrypted into the ciphertext. By contrast, the value of m is necessarily limited, one could not practically encrypt a prohibitvely long message. So, say, that for $n \to \infty$ we have $m/n \to 0$. And hence in that case the entropic advantage gained by the cryptanalyst will be very high, which means that in that case the UTC has marginal benefit, especially compared to the equivocation and entropy protection provided by Vernam's One-Time-Pad. Alas, for limited values of n, the UTC user may choose $m \to n$, which will zero out the entropic advantage gained by the cryptanalyst, and for that case UTC provides comparable advantage to Vernam.

In the general case the n plausible messages dictated by the cryptographic battleground are of various, not necessarily equal probabilities. Albeit, this makes little difference. The UTC user, having decided which of the n plausible messages to encrypt, will then select the additional (decoy) (m-1) messages from the most plausible down, so that the total measure of entropy assembled by the m messages will be maximized, and so will be the entropic advantage for the UTC user.

3.0 Implementation Notes

Some important practical challenges facing the implementer of a UTC are:

- Selecting the (m-1) decoy messages
- Preparing the plaintext for transposition
- Embedding the UTC within a larger cryptographic protocol

Ideally all these challenges will be met via an automated procedure, enhancing the practical attraction of the UTC protection.

3.1 Selecting the Decoy Messages

Normally a cryptographic practitioner will feed her secret message to the cipher of her choice, and generate the secure ciphertext. The UTC practitioner must engage in a preparatory action before so doing: she must combine her secret message with decoy messages. So she must first identify them, and then combine them properly.

The more serious the battleground, the more studied are the various options, and their corresponding messages. Alas, for casual circumstances the UTC practitioner may not have analyzed what other messages could have been sent, and some form of generic process is called for. This can be done manually, or through use of artificial intelligence. Given the secret message that needs protection, a modern AI inference engine will compose distinct messages with a rather different indication. For example: let the protected message be: (illustration 1).

George will meet Nancy at the bus station, before robbing the bank

A simple AI engine (or through manual generation) will build assorted decoy like:
George will meet Lucy in the bank
Harry and Susan will meet at the train station

In this illustration the decoys seem less plausible than the protected messages because their meaning is more lame. However, one could choose decoys like:
James decided not commit the fraud he was thinking about
The bank is well guarded, let's call the whole thing off

share roughly a par plausibility with the protected message.

In many instances the decoys are obvious, see illustration 2:

Let the protected message be: *We shall attack from the north on Thursday*

Natural decoys will be:

We shall attack from the south on Wednesday
We Shall attack from the east on Sunday

etc. Using a combination of four directions (north, south, east, west), and seven days of the week (Sunday... Saturday) one can construct 27 = 7*4-1 decoys. And by adding the world "not" after "shall" the number rises to 55 = 7*4*2 -1 decoys.

Once the decoys are selected, the challenge becomes how to assemble them with the protected message to construct the pre-transposition plaintext.

3.2 Preparing the Plaintext for Transposition

The UTC practitioner once she selected (m-1) decoys to guard her protected message will face the challenge to assemble it all into the pre-transposition plaintext. The basic way to do it, is straight forward: Let M be the protected message, and let d_1, d_2,....d_{m-1} be the (m-1) decoys. One would then compose the pre-transposition plaintext (PTP) as follows:

$$PTP = M * d_1\ d_2\ ....\ d_{m-1}$$

where the message and its decoys are simply concatenated, and where the asterisk (*) denotes any (optional) chosen separator so that the intended reader will know to dismiss and disregard everything written right of this separator (however long it is), and focus on the protected message M. The cryptanalyst will see before him all the *m!* permutations where each of the decoys, in turn, is written first, and becomes the suspected M.

Of course the assembly of the m messages may be exercised in more sophisticated manners than simply concatenation, but concatenation will work fine.

In this basic implementation the PTP is identified as a list comprised of m elements (the protected message M and the m-1 decoys), which can be mixed into m! distinct permutations. The length of the PTP is on average m times the length of the protected message M. An alternative way would be to regard the PTP as comprised of w words, which get mixed. This may help reduce the size of the PTP.

Let's check again illustration 2: where:

M = "We shall attack from the north on Thursday"

The 55 decoys: d_1, d_2,....d_{55} could be written explicitly one after the other, but also as follows:

*PTP = "We shall attack from the north on Thursday * south east west Sunday Monday Tuesday Wednesday Friday Saturday not"*

again, provided that the PTP will be regarded as a list of 19 words to be mixed up. The gain here is that the PTP is only about twice the size of M (while packing 55 decoys!).

The UTC practitioner could also apply non-uniform definition of transposition elements. Namely one could parcel the PTP to transposition elements of various sizes: be it a full paragraph, a sentence, a word, or a letter, or even a bit. For example: (illustration 3) let the protected message be:

M = "The password for account number 45678291 is JhGn8817%E"

One may be build it into a PTP like this:

*PTP = "The password for account number 45678291 is JhGn8817%E * 8976thbbgf$@!988764654439RE1@0"*

which may be parceled as follows:

*PTP = "The_password_for_account_number 4 5 6 7 8 2 9 1 is J h G n 8 8 1 7 % E * 8 9 7 6 t h b b g f $ @ ! 9 8 8 7 6 4 6 544 39 RE1 @ 0"*

Here the space is used as the delimiter that separates the transposition elements. Come to think about it, the rules for how to parcel the PTP may be part of the shared secret.

3.3 Embedding the UTC within a larger Cryptographic Protocol

Consider a battleground where a party faces a binary decisions, say (i) "to go north" (N) or (ii) to go south (S). The party chooses to go north, N, and so uses UTC to encrypt the PTP: NS. The intended readers of the message will know how to interpret it (will realize that N is the message and S is the decoy), but the cryptanalyst will remain clueless, and as ignorant as before capturing this message. In this case the two messages can be written in the clear, It makes no difference. Indeed the UTC works with messages written in the clear, with no further encryption.

In general though, the exposure from posting the decoys and the message in the clear is risky because it tells one's enemies how insightful the UTC user is, how she discerns the various options. It is therefore advisable to embed the UTC within a larger cryptographic context. This can be done (i) a-priori: prior to the transposition, or (ii) post-priori: following the transposition.

A priori: the UTC user determines his message M and his decoys d_1, d_2, ...d_{m-1}. He then applies his favorite regular intractability cipher to encrypt the m messages before assembling the PTP. The intended reader would first "un-mix" the cipher to the PTP, discard the decoys, then decrypt M to its plaintext form and read it plainly.

Post_priori: The UTC cipher will be encrypted using any common intractability cipher. The intended reader will first decrypt the intractability cipher, then be presented with the UTC cipher and extract from it the protected message M.

Obviously a UTC user could apply both a-priori and post-priori measures using the same or different cipher. This guarantees that by adding the UTC one in no way distract from his otherwise security, it only helps!

4.0 Security

Unlike intractability based ciphers which operate under the shadow and fear of a well concealed compromise, the UTC is safe in as much as the protected message M and the (m-1) decoys d_1, d_2,.....d_{m-1} all do have a UTC key that encrypts them to the given cipher, hence the cipher projects *mathematical parity* towards the m candidates for M.

This broad statement does come with an important qualifier. It is necessary that the computational burden for the UTC (encryption and decryption) is always limited. If for some keys the computation burden is prohibitive then such keys may be discounted and the cipher is no longer the Ultimate Transposition Cipher.

What about a re-use of a given UTC key? If the same UTC key is applied to q messages then security depends on whether the UTC is applied message-wise or word-wise, as explained below:

Message Wise UTC use: in this case a protected message M and the (m-1) decoys d_1, d_2,.... d_{m-1} are regarded as the m elements to be transposed. The respective transportation key space is of size: $|K_{transposition}| = m!$

Word-Wise UTC use: in this case M and the decoys are divided into units smaller than the full message, to be regarded as words. If a PTP message is divided on average to w words each, then the transportation list is comprised of *mw* elements, and the respective UTC key space is of size $|K_{transposition}| = (mw)!$

If the same UTC key is used without a change to encrypt q messages encrypted message-wise then from an analytic standpoint the system remains robust because any UTC key used will result in a plausible message candidate pointed out for every encrypted message. The q ciphertexts do not provide any information to sort these candidates out, and hence much as any UTC key will point to one of the m candidates as the protected message, M, so any UTC key will point to q plausible message candidates for the q messages, and the q ciphertexts will not be helpful in sorting out the options.

If the messages are encrypted word-wise then a given key which would point to a given message candidate for one message might point to a meaningless permutation of the *mw* words for the next message, and thus will be discounted. So, as q and w increase the number of UTC key candidates that would 'remain standing' is coming down, since in general there are more grammatically senseless sequences of the words than there are meaningful ones.

What if over time, events suggest the true M for, say, the first message? This will be harmful, but in a limited way because for a message-wise implementation there are (m-1)! UTC keys that all point to the true M. The cryptanalyst will not know which of those possible UTC keys have been used. Alas, if circumstances keep revealing the second message, the third, etc, then there remain less and less UTC keys that would point to all these known M messages, and the reduced key space will bring the security down.

Because of that danger one may choose to use the UTC entropy to conceal in a UTC message the key for the next UTC message:

$$UTC_{Ki}(M_i + D_i + K_{i+1}) = C_i$$

where D_i represents the totality of decoys used for message i=1,2,....q-1, and C_i is the ciphertext for message i; K_i is the key used for message i, and K_{i+1} is the key used for message (i+1).

If M is a random sequence, then the (m-1) decoys may be random too. And in that case all the n! keys will be equally plausible (where $n \geq m$ is the number of transposed elements in the PTP). Even for moderate n values n! is a very large number. This point suggests that M and the decoys may be nominally encrypted before applying the UTC (so they all look randomized). The same point also suggests applying UTC after encrypting the UTC transposed permutation (UTP).

This reasoning leads to a combined protocol where the protected message M is first turned into a pre-transposed plaintext (PTP), which is UTC transposed to UTP. Then the UTP is encrypted with any common cipher, to generate the encrypted-UTP: eUTP. The eUTP in turn, will be arbitrarily parceled into w words, which will be UTC transposed (key space: $|K_{eUTP, w}|= w!$, resulting in a UTC transposed eUTP (UeUTP). The UeUTP may sustain another UTC round where the UeUTP is parceled out to w' words, which are UTC processed to generate UUeUTP. And again, as desired, building at will intractability protection hinged on the built-in equivocation of each UTC round.

5.0 A Working UTC Example

A working example of an ultimate transposition cipher is cited in reference 3. A short description follows:

We first describe a simple transposition mechanism of "migration" where elements of the original permutation are migrated to a new permutation one by one, "knocked out" by cyclical counting of k elements. Specifically: let the ordered numbers: P_0 = 1,2,......n represent a given permutation of n elements. This permutation will be processed to another permutation P_k based on a natural number k, as follows: Begin counting the P_0 series of numbers such that when its end is reached, the counting continues from the other end. The counting stops on element R ($1 \leq R \leq n$). Remove R from P_0 and position it as the next element in P_k. Since at this point P_k is empty, R then becomes the first element in P_k. Continue counting (either in the same direction, or in the opposite direction, as the case may be) starting from the element in P_0 which is placed next to R. The count of k elements will stop upon element S in P_0 ($1 \leq S \leq n$; $S \neq R$). Remove S and place it next in line in P_k. P_k will now look as P_k = RS. Continue in this manner, each round removes one element from P_0 to P_k, until P_0 is empty and P_k contains the numbers 1,2,.... n in a different permutation. We may write then:

$$P_k = E_k (P_0)$$

where E_k is this transposition-encryption, based on key k. For example: let P_0 = 0123456789, applying migration we write: P_{11} = 0259784136; P_{69} =8407291635; P_3 =2581607493

It can be shown that this transposition procedure has a key space of $n_!$ (sub-factorial) defined as:

$$n_! = \prod P_i^{n_i}$$

where P_i is the i-th prime number, and n_i is the power to raise Pi such that:

$$P_i^{n_i} \leq n \;\; and \; P_i^{n_i+1} > n$$

Since n_i < n!, this migration transposition cipher is not complete, not ultimate. To render it 'ultimate' one would '*ghost dress*' the 1,2,..n permutation, namely: add, say, g 'dummy numbers' (ghosts) after each number:

$$GhostPTP = 1{*}{*}{*}...{*}2{*}{*}{*}...{*}3{*}{*}{*}...{*}4{*}{*}{*}...{*}n{*}{*}{*}...{*}$$

GhostPTP is a permutation list comprising *ng* elements. One chooses the value of g such that:

$$(ng)_i > n!$$

As shown in reference 3, by applying the migration procedure to the 'ghost dressed' permutation, followed by scrubbing all the ghosts, one secures an ultimate transposition cipher.

6.0 Reference

1. Bansal 2011 "Transposition Technique for Cryptography" IUP Journal of Computer Sciences Satish Bansal and Rajesh Shrivastava

2. Malik, S. 2011 ; "A Novel Key-Based Transposition Scheme for Text Encryption" Frontiers of Information Technology (FIT), 2011 Dept. of Comput., Nat. Univ. of Sci. & Technol., Islamabad, Pakistan

3. Samid, G. 2015 "Equivoe-T: Transposition Equivocation Cryptography" IACR ePrint Archive https://eprint.iacr.org/2015/510

4. Sokouti 2009, "An Approach in Improving Transposition Cipher System", Sokouti M, Sokouti B and Pashazadeh S. Indian Journal of Science and Technology, Vol. 2, No. 8, pp. 9-15.

5. Toemeh 2007 "Breaking Transposition Cipher with Genetic Algorithm" ELECTRONICS AND ELECTRICAL ENGINEERING Toemeh, Arumugam

6. Vernam 1918, Gilbert S. Vernam, US Patent 1310719, 1918.