A Randomized High-Security Cipher Combining Deniability with Pencil-and-Paper Decryption

Grant D. Schultz

Abstract: With key logging trojans and other malware in the wild, and operating systems being too complex to protect against government surveillance, it is necessary to have high-security ciphers which can be used without the aid of a computer. Some proposals in this area include Solitaire [1] and Handycipher [2]. The cipher described here further explores this space by introducing deniability.

I. Introduction

Today's operating systems are sufficiently complex and inscrutable that only organizations with vast software and hardware resources can ensure that their computer systems are entirely free of malware. All other users have no guarantee that their every keystroke is not being logged. This implies that virtually no computer can be trusted for the highest security needs.

Computers disconnected from all networks are somewhat less vulnerable, but are still susceptible to having their electronic or audible emissions picked up by nearby monitoring devices, or having key logging devices physically installed into them. Seizure and post hoc forensics are also a threat.

For large amounts of data, only computers will be able to perform encryption for us, and we shall have to take our chances. But small amounts of data such as passwords, complex answers to security questions, GPS coordinates, account numbers and short messages can also benefit from encryption.

Since the advent of computers, pencil-and-paper ciphers have received decreasing attention from experts, as it was assumed that no such algorithms could remain secure against adversaries with large computing resources. However, the above considerations warrant a renewed search for secure manual methods. Several recent proposals have shown that such methods may offer serious security for low-volume use. Schneier's Solitaire [1] is a stream cipher combined with an innocuous way to store the key. Kallick's Handycipher [2] employs randomized encryption to achieve a one-to-many mapping from plain to ciphertext. This paper explores a different area in the space of low-volume, high-security ciphers, namely the incorporation of deniability.

II. Background

The one-time pad has been in use since the early twentieth century, is simple to use, and provides perfect security, but must have unique keys the same length as the total of all messages to be encrypted. Modern computer ciphers allow reuse of the key because the encryption transformations are sufficiently complex to obscure the key. To obtain a similar obscuring of the key, the cipher proposed here relies on randomization.

Homophonic Ciphers

A homophonic cipher maps each symbol p in the plaintext alphabet P onto one of many ciphertext symbols x, which are elements of X, where $|X| > |P|$. X is divided into $|P|$ disjoint subsets, $\{X_1, X_2, \ldots, X_{|P|}\}$, one subset per plaintext symbol. To encrypt a plaintext symbol p, a random element of subset X_p is chosen to represent it. The key to such a cipher is the mapping from plaintext to ciphertext, or the rule from which the mapping can be reconstructed.

Rivest's Chaffing-and-Winnowing, and a Variation

Chaffing and Winnowing [3] uses a message authentication code (MAC) to separate a received stream of packets (or series of numbers) into "wheat" (those packets whose MACs are valid) versus "chaff" (those whose MACs are invalid). The packets whose MACs are valid form or otherwise indicate the intended message. This achieves confidentiality without the explicit use of a true encryption algorithm.

A variation on this approach is as follows: For a particular plaintext symbol p, a random number r is generated from a large space. The random number is combined with a key K via a hash algorithm $\text{HASH}_K(r)$, which produces a number in the range $1 \ldots |P|$. If it hashes to p, then the random number becomes the ciphertext equivalent of p. Otherwise, r is discarded, and the above procedure is repeated until a random number is found that does hash to the desired plaintext symbol. (This will take on average $|P| / 2$ tries, unless a dictionary of the random numbers and their plaintext equivalents is maintained.) This process is repeated for each plaintext symbol in the message. The ciphertext is thus a series of semi-random numbers, which are homophones for the corresponding plaintext symbols.

Under the above approach, the number of ciphertext equivalents for a given plaintext symbol is practically unlimited. The ciphertext numbers generated by this process are no longer completely random, as the generating process must sift through many random numbers to find a particular output. However the decrease in entropy can be made arbitrarily small by enlarging $|C|$.

Deniability

To insert deniability into such a scheme, consider a series of n identical length plaintexts, the first genuine, and the rest decoys: $P_1, P_2, \ldots, P_n$. Each of these begins with a particular (possibly different) character,

$P_1 = p_{11}, \ldots$
$P_2 = p_{21}, \ldots$
.

.
$P_n = p_{n1},...$

There is a corresponding set of keys, one per plaintext: K_1 is the real key, and $K_2,...K_n$ are decoys, which can be revealed if necessary.

To encrypt the p_{*1} characters simultaneously, we modify the above approach to search for a random number r that simultaneously hashes to each desired plaintext character under the respective key:

$\text{HASH}_{K1}(r) \text{ -> } p_{11}$
$\text{HASH}_{K2}(r) \text{ -> } p_{21}$
.
.
$\text{HASH}_{Kn}(r) \text{ -> } p_{n1}$

Once we find such an r, it becomes the desired ciphertext symbol, c_i. This will require searching on average $|P|\text{^}n$ / 2 random numbers to find an r which produces the right outputs. This process is then repeated for the remaining sets of plaintext symbols p_{*i}, to give the complete ciphertext for the message.

Decryption merely requires evaluating the hash algorithm with the genuine key, once per ciphertext symbol.

III. Details of the Proposed Scheme

The hash algorithm described in this section relies on simple mathematical functions that can be computed by hand. Pseudocode for retrieving one plaintext character from one ciphertext value c with key K is as follows:

```
Add K and c with carry to give the sum Q;
Repeat the following until all digits of Q have been consumed:
{
      Consume the next digit from Q, and use it to select
            a formula from Table 1.
      Consume digits from Q to fill in the formula; re-use
            digits from Q by looping around if necessary; i.e.,
            if you start a formula, finish it.
      Evaluate the formula.
      Take the indicated digits from the result, and
            insert them into the summing grid.
}
Add the numbers in the summing grid with carry.
Take the resulting sum modulo |P|.
```

The following table gives the formulas referenced above:

Table 1, Formulas

Q digit	Formula -> output	
0	(q * q) + qq	-> idd
1	((qq + q1) * (q + 3)) + 9	-> iddd
2	qq + qq	-> idd
3	(q + q7) * 3q + q + 1	-> iddd
4	((q + 1) * q) + (q * (q + 1))	-> idd
5	q + ((q + 1) * (q + 1)) + q	-> idd
6	((q + qq) * 2q) + 8	-> iddd
7	((q + q) * (q + 1)) + q + 7	-> idd
8	qq + q + 3	-> idd
9	((qq + 1) * (q + q)) + 9	-> iddd

where

> adjacency indicates concatenation, including where digits are used
> q is a digit from *Q*
> i is a possible resulting digit that should be ignored
> d is a resulting digit that is to be copied to the summing grid

So if Q begins 11584... then the algorithm selects formula $\underline{1}$, and evaluates (($\underline{15}$ + $\underline{8}$1) * ($\underline{4}$ + 3)) + 9, giving 681.

If the result of evaluating a formula does not produce enough digits to match the number of d's specified, then the result should be zero-filled on the left. A digit used to select a formula is not re-used in the formula itself.

The summing grid has 3 columns, and as many rows as are needed to hold the numbers produced. It should be filled in from top-to-bottom, left-to-right.

The key and ciphertext are first combined, so that the selection of formulas is controlled by both. This ensures that a different series of formulas will be chosen for each ciphertext number, while preventing the actual number used from being available to an attacker. (If only *C* were used, the formulas would still vary, but they would be known. If only *K* were used, the formulas would be the same for each ciphertext number.)

Example

As an example, assume a 37-symbol plaintext alphabet, with 'A'=0, 'B'=1...,'Z'=25, '0' (zero)=26, '1'=27,..., '9'=35, <space>=36, and the following plaintexts:

1. `SWISSBNK 6448511AV`

2. `FACEBK PASSWEIRD12`
3. `TWITTR ADFGVX12345`

With the following keys,

1. `24051308142361415759681324934867`
2. `34067961028221011552160270760697`
3. `56116290901116162757605812392787`

the following ciphertext might result:

```
97193968311299338617852937305237
26958466583056425697240517151760
16506310653271750366954187101004
94616701976229795397502103685220
30744650133231284955444773253752
86354582878388333461387646035707
78407540186770670109066494651598
94337252134952739647847762355982
69945053548249686985575450924889
82116118958146866611167207869607
35466643913273093347537473131605
24721463022817198258314778533870
70113126875206002755029071851137
78647987352806358678698420317881
47863997354311850578447878676720
69813741185206814504193038844038
06382355469988458540937111525189
34643127364515430750630378839812
```

During decryption with key 1, the summing grid for the first ciphertext number would look like this:

```
303
625
250
495
280
84-
```

which sums to 2793. Reducing that modulo 37 yields 18, which is the numeric equivalent to 'S', the first letter of plaintext 1.

IV. Practical Considerations

Due to the asymmetry of effort between encryption and decryption, encryption will need to be performed by computer. This will also ensure sufficient randomness in the ciphertexts. Use of a computer represents a security risk, so the computer should be a physically secure system not connected to a network, which can be wiped when encryption is finished. Because the algorithm is simple, a small embedded computer (e.g., an Arduino) with any networking capability disabled could also be used to implement encryption.

The decoy messages should be harmless to the sender and recipient, yet of a nature that would deflect suspicion if they have to be revealed. It is advisable to include as many decoy messages as practical, because an attacker will suspect that more remain hidden. Decoy messages might include genuine account numbers or passwords to websites that could be safely revealed with little consequence.

The genuine key should be at least 32 decimal digits. Memorizing such a key is difficult, so a scheme such as described in [4] can be used to store the key(s). The decoy keys can be shorter than the real one, to increase the likelihood that a brute-force search will find them first.

By varying the modulus used to reduce the final sum, the system can accommodate digits, nulls, punctuation or word separators as desired. To make the messages equal length, nulls should be inserted into the shorter messages at random locations.

Tests show that one plaintext character can be retrieved in around 15 minutes by hand. Thus a message of 24 characters could be retrieved in around 6 hours by one person. Because each ciphertext number is independent of all others, errors in decrypting do not propagate.

As with Handycipher, a session key could be incorporated into the algorithm, although this has not been explored.

V. Analysis

Traditional attacks on homophonic ciphers depend on eventual repetition of ciphertext values. With ciphertext values of 32 digits, no c is expected to repeat across all ciphertexts for any particular key. Therefore, methods of cryptanalysis that use frequency information cannot be used here.

Brute-forcing all possible K will give many plausible candidate plaintexts, leaving the cryptanalyst with no way to distinguish the genuine message from others resembling plaintext (or from the decoy messages). Thus a ciphertext-only attack is not viable.

The most advantageous position from which to launch a known-plaintext attack is to assume that both the genuine and decoy plaintexts are known. But to work backwards from the plaintexts to the sums to the entries in the summing grid implies knowing the entire key or somehow solving for it. This also does not appear to be feasible.

VI. Conclusion

We have presented a high-security, hand-decipherable system that encrypts multiple messages with an equal number of keys. This provides the opportunity to reveal one or more decoy pairs while keeping a specific pair secret.

With hackers and governments undermining the security of nearly all computing devices, more effort is needed to explore the neglected area of low-volume pencil-and-paper cryptography.

References

[1] B. Schneier, "The Solitaire Encryption Algorithm," available at https://www.schneier.com/solitaire.html.

[2] B. Kallick, "Handycipher: a Low-tech, Randomized, Symmetric-key Cryptosystem," available at https://eprint.iacr.org/2014/257.pdf.

[3] Rivest, Ron, "Chaffing and Winnowing: Confidentiality without Encryption", available at http://people.csail.mit.edu/rivest/Chaffing.txt.

[4] C. Ellison, C. Hall, R. Milbert, B. Schneier, "Protecting Secret Keys with Personal Entropy", available at http://www.schneier.com/paper-personal-entropy.html.