http://ansinet.com/itj

ITJ

ISSN 1812-5638

INFORMATION TECHNOLOGY JOURNAL

ANSInet
Asian Network for Scientific Information
308 Lasani Town, Sargodha Road, Faisalabad - Pakistan

Information Technology Journal 4 (1): 87-95, 2005
ISSN 1812-5638

Implementation of One-Time Pad Cryptography

K.V.O. Rabah
Department of Physics, Eastern Mediterranean University
Gazimagusa, North Cyprus, via Mersin 10, Turkey

Abstract: In this study we show how a moderate amount of random key stream can be used to generate a lifetime supply of keys for one-time pads. We also show how arbitrarily many parties can correspond using the same random key, without compromising one another's communications. The net effect is to make the unbreakable one-time pad practical for widespread use. Provided also is the practical information that you can use to set up your own one-time pad encryption system. The study further describes subtle refinements that you won't find in other books or articles.

Key words: Cryptography, cryptology, one-time pad, data security, autokey cipher, key enhancement

INTRODUCTION

Secrecy is the heart of cryptography. Encryption is a practical means to achieve information secrecy. Modern encryption techniques are mathematical transformations (algorithms) which treat messages as numbers or algebraic elements in a space and transform them between a region of "meaningful messages" or cleartext and a region of "unintelligible messages" or ciphertext. In order to restore information, an encryption transformation must be reversible and the reversing transformation is called decryption. Conventional, encryption and decryption algorithms are parameterized by cryptographic keys. An encryption algorithm and a decryption algorithm plus the description on the format of messages and keys form a cryptographic systems or a cryptosystem[1,2].

Currently, there are two popular kinds of cryptographic protocols: symmetric-key and asymmetric-key cryptosystems. In the symmetric-key protocols a common key (i.e., the secret-key) is used by both communicating partners to encrypt and decrypt messages. In the public-key protocols there are two associated keys; one is kept private by the owner and used either for decryption (confidentiality) or encryption (signature) of messages. The other key is published in the public domain (public-key server) to be used for the reverse operation or decryption[1,3].

A bit of history-Cryptography as a science was originally developed by the Arabs. The year 1412 saw the publication of Subh al-a 'sha, a 14-volume encyclopedia written by Shihab al-Din al-Qalqashandi. The text described transposition and substitution ciphers[1]. The Arabs were light-years ahead of the Europeans because their mathematics was more advanced - and cryptography relies heavily on math. While the Europeans were still struggling with Roman Numerals, the Arabs had already discovered the principle of zero. The word cipher is derived from the Arab word 'al cifr', literally meaning nothing or zero.

According to Shannon a cryptosystem can be characterized with the following properties; which states that the ciphertext message space is the space of all possible messages while the cleartext (human readable) message space is a sparse region inside the message space, in which messages have a certain fairly simple statistical structure, i.e., they are meaningful; a (good) encryption algorithms is a mixing-transformation which distributes the meaningful messages from the sparse and meaningful region fairly uniformly over the entire messages space. While, Kerchoffs; in 1883 listed the requirements to design cryptosystems, today referred to as Kerchoff's principle. This principle states that the security of the system has to be based on the assumption that the "enemy" has full knowledge of the design and implementation details of the crypto-algorithm. The only missing information for the "enemy" is a short easily exchangeable random number sequence, the secret-key[1]. Without this secret-key, the "enemy" should not have the chance to even suspect that on an observed communication channel, hidden communication is taking place[2]. Since the adversary may obtain this information eventually, it is preferable not to rely on its secrecy when assessing cryptographic strength. Most of the modern crypto-software in use today meets this principle. Combining these two thinking gives rise to a good cryptosystems design and implementation.

To date we have not been very successful in relating, via computation, between two pieces of information if one of them merely "looks random" while in fact they are completely dependent on one another (for example,

plaintext, ciphertext messages in many cryptosystems). As a result, modern cryptography has its security based on a so-called complexity-theoretic model. Security of such cryptosystems is conditional on various assumptions that certain problems are intractable. Here, "intractable" means that the widely available computational methods cannot effectively handle these problems in a reasonable time frame.

If random variable follows the distribution and is independent, from any given information, then there is no way to relate a uniformly random variable to any other information by any means of "computation." This is exactly the security behind the only unconditionally (or information-theoretically) secure encryption scheme: one-time pad, that is, mixing a uniform random string (called key string) with a message string in a bit-by-bit fashion. The need for independence between the key string and the message string requires the two strings to have the same length. Unfortunately, this poses an almost unpassable limitation for a popular and practical use of the one-time pad encryption scheme.

For simplicity let's have a look at how randomized key can be achieved. Random key generation can simply be achieved via use of permutation. Permutation technique involves rearrangement of the characters of a given plaintext message to convert the message into an anagram that looks like a message with random characters. For example, here is a permutation of {1,2,3,4,5} to {4,5,1,2,3}, which can be used to transform (randomize) a string of five characters by moving each character from its original position to the position defined by the permutation. For example, the string "APPLE" would be translated to the string "PLEAP". What about reversing back the text? Well permutation has an inverse, another permutation that does the exact opposite of the original permutation. For example, the permutation {4,5,1,2,3} has an inverse {3,4,5,1,2} which should unscramble the string "PLEAP" to get back the original string "APPLE". In real-time application, for example, most messages consist of 7-bit ASCII characters or 8-bit in hexadecimal which we can represent with integer numbering (01234567) according to Table 1, to give permutation representation e.g., {7,5,4,6,3,0,2,1}. By scrambling the bits to create a random set of bits (using inverse of our permutation i.e., {5,7,6,4,2,1,3,0} you can get the desired encryption. For example, an 8-bit character, the byte/character 'G', which have position 71 into ASCII character set and have the binary representation '0100 0111' can be encrypted/decrypted as shown in Table 1.

Permutation techniques are usually used in conjunction with other techniques such as substitution and encryption-functions like XOR. Other functions such as binary addition, multiplication and modular arithmetic functions are also common. The goal of this study was to show how the one-time pad can be accomplished by a combination of these techniques.

Table 1 An example of permutation algorithm for encryption/decryption.

	byte/character 'G'	Permutation {7,5,4,6,3,0,2,1}	Inverse Permutation {5,7,6,4,2,1,3,0}
Binary	0100 0111	1110 0100	0100 0111
Decimal	71	228	71
Hex	47	E4	47

One Time Pad-Background: The term "one-time pad" refers to any method of encryption where each byte of the plaintext is encrypted using one byte of the key stream and each key byte is used one time then never used again - and its is the only absolutely secure cipher in use today. The one-time pad system itself was perfected in 1917 during the First World War[4,5]. Random keys were written on sheets of paper that were glued together to form a pad. Each key was used only once-hence the name, one-time pad. The key stream for a one-time pad must be a true-random stream, meaning that every key byte can take any of the values 0 to 255 with equal likelihood and independently of the values of all other key bytes. By contrast, in a pseudo-random key stream the value of each byte after the first several is mathematically derived from the values of a few preceding bytes.

The practical difficulty of using a one-time pad is that the key bytes cannot be reused. This means that even for a two-way exchange of messages, each party must have a sufficient supply of key material on hand so that they cannot run out before more key stream can be furnished. For N-way exchanges, the amount of key required increases quadratically. Many papers on cryptography assume that the only possible way out is for the key to be generated by some mathematical algorithm and then expend a great deal of cleverness in finding an algorithm that adequately approximates the properties of a true-random key. For more detailed information, one may refer to ref.[6] which gives a broad survey of such methods. The weakness of such methods is that determining a portion of the key stream will allow an opponent to reconstruct the entire key stream using the same mathematical algorithm. In this paper we will try to present a solution intermediate between a pure random key and a pure mathematical generator. However, be warned that the ultimate crypto-security depends on how cleverness you can keep your key secure from your enemy!

Why is One-Time Pad Perfectly Secure?: One-time pad encryption algorithm can be generalized using the following equation:

$$Ci=E(P_i, K_i) \text{ for } I=1,2,3,....,n$$

where, E is the encryption operation, P_i is the I-th character of the plaintext, K_i is the I-th byte of the key used for this particular message, C_i is the I-th character of the resulting ciphertext and n is the length of the key stream. Both the key stream K and the encryption operation E must be kept secret. The key for each individual message is the starting location in the entire random key stream used for this encryption as will be seen later in the text. Although the term byte has been used and will continue to be used for each unit of the key stream, it is not necessary that each unit of the key be 8 bits long. In fact, a key with larger units gives better protection against a known-plaintext attack, since then there may be several values of the key which produce the given ciphertext byte from the known plaintext byte. Similarly, the plaintext could be encrypted in units other than 8-bit bytes. For present purposes, however, encryption based on bytes is sufficiently general.

For practical purposes, the key for a one-time pad cipher is a string of random bits, usually generated by a Cryptographically Strong Pseudo-Random Number Generator (CSPRNG)[7]. However, for ultimate security, it is advisable to generate the key using the natural randomness of quantum mechanical events (such as those detected by a Geiger counter as used in Experimental Physics), since quantum events are believed scientifically to be the only source of truly random information in the universe. One-time pads that use CSPRNGs are open to attacks, which attempt to compute part or the entire key. If the key is truly random, an XOR-based one-time pad is perfectly secure against ciphertext-only cryptanalysis[1]. To encrypt plaintext, P, with a key, K, producing ciphertext, C, simply compute the bitwise exclusive-or of the key and the plaintext: C=K^P. To decrypt ciphertext, C, the recipient computes: P=K^C. It is that simple and it's perfectly secure, as long as the key is random and is not compromised. A simple example of one-time pad encryption/decryption algorithm, implemented mod 26, goes like this:

Encryption: CLAUDIUSDIDIT+SYQJOWJQGBOEF
=VKRESFEJKKSNZ (L:12+Y:25 ->K:37)
Decryption: VKRESFEJKKSNZ-SYQJOWJQGBOEF
=CLAUDIUSDIDIT

This means an attacker can't compute the plaintext from the ciphertext without knowledge of the key, even via a brute force search of the space of all keys! Trying all possible keys doesn't help you at all, because all possible plaintexts are equally likely decryptions of the ciphertext. This result is true regardless of how few bits the key has or how much you know about the structure of the plaintext. To see this, suppose you intercept a very small, 8-bit (1-byte), ciphertext. You know it is either the ASCII character 'S' or the ASCII character 'A' encrypted with a one-time pad. You also know that if it's 'S', the enemy will attack by sea and if it's 'A', the enemy will attack by air. That's a lot to know. All you are missing is the key, a silly little 8-bit one-time pad. You assign top your crack staff of cryptanalysts to try all 256, 8-bit one-time pads. This is a brute-force search of the keyspace. The results of the brute force search of the keyspace is that your staff finds one 8-bit key that decrypts the ciphertext to 'S' and one that decrypts it to 'A' and, you still don't know which one is the actual plaintext.

How to Use One-time Pads for Secret Communications: One-time pad cryptosystem is known to be the only cipher system that cannot be cracked by any of the National Security Agency (NSA) in existence-or by anyone else for that matter. It is well a known fact that a message encrypted using a one-time pad cannot be broken because of two facts: the encryption key is a random number and, the key is used only once.

One time pad is known to be a proven system. Intelligence agencies routinely use many different kinds of encryption systems-ranging from mechanical devices to invisible inks to computer software[2]- but for mission critical messages that must be 100% secure they always use a one-time pad. At the height of the cold war during the fifties and sixties, Soviet spies in the USA used one-time pads to communicate with their controllers, usually located inside Russian embassies and consulates. Not a single message was cracked by the NSA to-date and, none of those messages ever will be cracked. The one-time pad system is still being used today by intelligence agencies in most developed and developing countries including the underworld and resistance movement across the world.

We provide practical information that you can use to set up your own one-time pad encryption system. The study describes subtle refinements that you won't find in other books or articles. After studying this article you will have all the necessary knowledge you need to set up a 100% secure system of communication that cannot be cracked by any NSA, or any other organization in any given time. If you're playing by Big Boys' Rules, the one-time pad will keep you out of the troubled world of eavesdropping in your privacy world.

Step 1-Create the Key: The core of the one-time pad system is the random key. A key is a block of numbers

(or bits if you are working in binary systems) that is used to transform your original message (the plaintext) into a coded message (the ciphertext). Before you can begin to work with a one-time pad system, you need to create a random key. Before you can create a random key, you need a method for converting alphabet characters into numbers. The chart below illustrates a workable system that is simple and easily remembered:

A	B	C	D	E	F	G	H	I	J	K	L	M
01	02	03	04	05	06	07	08	09	10	11	12	13

N	O	P	Q	R	S	T	U	V	W	X	Y	Z
14	15	16	17	18	19	20	21	22	23	24	25	26

National security agencies (NSA) usually use codebooks containing often-used words and phrases that are represented by numbers. For example, rather than encrypting a phrase like "safe house A8" to 1917 2876 1432 2104 1531, the coding clerk might simply use 2029. Spies and agents, on the other hand, cannot afford to carry incriminating evidence like bulky codebooks, so they use instead the simplified conversion method shown below and spell out every word in full.

Now you're ready to create a key. First, write down a series of random alphabet characters, such as: KRTOWDBRZEJSPWSHLM. To make the key easier to work with, break it into blocks of two characters: each, thus: KR TO WD BR ZE JS PW SH LM.

Now use the conversion table shown above to convert the alphabet characters into numbers. For example K=11 and R=18, so the first block KR becomes 1118. The result is: **1118** 2015 2304 0218 2605 1019 1623 1908 1213.

You've just created your first one-time pad. This is also called the key. (Normally you would create a much longer key than this, enabling you to send a number of messages before the key is used up.) As you use the blocks of numbers to encrypt messages, you would cross out each block you've used. This will ensure that you use a block only once. (We'll simulate crossing out a block by bolding it. Here the first key string 1118 is bold-out):

1118 2015 2304 0218 2605 1019 1623 1908 1213

You would normally create two copies of the key and provide one copy to your intended recipient or communicating partner.

Step 2: Format your Message: Suppose that the message you want to send is: **HIGHLY TOP SECRET**. You would next format your message into blocks of two characters each, yielding HT OP SE CR ET. Next, use the conversion chart above to convert the alphabet characters into numbers. In the example we're using H=08 and T=20, so the first block would be 0820. The entire string becomes: 0820 1516 1905 0318 0520. You can now see how using blocks makes the text increasingly difficult for anyone to crack, even at this stage.

Step 3: Encrypt your Message: We need some way to indicate to our recipient where the key begins, otherwise he/she won't be able to decrypt. Remember in our earlier example, we created a key and stroked off (in bold) the blocks we have already used. Here's what our key now looks like: **1118** 2015 2304 0218 2605 1019 1623 1908 1213.

The first four-digit group is a pointer to the starting place in the one-time pad. That is, the starting position in the key is at block 2015. So we'll place the string 2015 at the beginning of our message so the recipient will know how to decrypt it. The 2015 plus the message of 0820 1516 1905 0318 0520 becomes 2015 0820 1516 1905 0318 0520 because we place the pointer 2015 at the beginning of the string.

We're now ready to encrypt. First we write out the plaintext. Then directly below it we write out the key. Then we add the key to the plaintext using Fibonacci addition. This means we do no carrying. For example, 7+6 would yield 3 not 13 and, 8 plus 4 would yield 2 not 12. Here's how our top spy, Nikita, working sheet would look like:

Plaintext	2015	0820	1516	1905	0318	0520
Key	--	2304	0218	2605	1019	1623
Ciphertext	2015	2124	1724	3500	1327	1143

The encrypted message 2015 2124 1724 3500 1327 1143 is ready to be sent to our recipient. Thereafter, we can sleep peacefully knowing that it cannot be cracked by anyone except the recipient, Michael. To decrypt the message, Michael simply reverses the calculations.

Step 4: Decrypting the Message: To decrypt a message, here is how we reverse the calculations. We subtract the key from the ciphertext using Fibonacci subtraction. This means we allow no negative numbers. We add 10 if required. For example, 3-7 would yield 6 (because we add 10 to 3 so that we're able to subtract 7 from 13). The first block in the ciphertext tells our recipient where to start in the key. Here's what Michael's working sheet looks like:

Ciphertext	2015	2124	1724	3500	1327	1143
Key	2015	2304	0218	2605	1019	1623
Plaintext	----	0820	1516	1905	0318	0520

Here's how we subtract 2304 from 2124:

- The first 2-2=0
- The second column is 1 - 3=8 (because 11 - 3=8).
- The third column is 2 - 0=2.
- The fourth column is 4 - 4=0.

Using the conversion chart described earlier, the recipient converts the string of numbers back into alphabet characters. In this example, 08=H and 20=T, so the first block 0820 converts to HT. The string 0820 1516 1905 0318 0520 becomes HT OP SE CR ET. The recipient reformats it to become HTOP SECRET. This argument is easily generalized to keys (and plaintexts) of arbitrary length and, in such case we need to design an algorithm, which we can easily use to generate our key on the fly.

Implementation Guidelines for Critical Message

Rule 1-Numbers: Spell out all numbers in full in your plaintext. For example, 274 becomes TWO SEVEN FOUR.

Rule 2-Negatives: Always add emphasis to the word NOT in your plaintext. For example, you would write AGENT NIKITA NOT RPT NOT AVAILABLE FOR MEETING WEDNESDAY, where RPT stands for REPEAT.

Rule 3-Punctuation: Use an X for each period in your plaintext. For example, MESSAGE RECEIVEDX SEND MORE INFOX. All other punctuation must be written out in full, e.g., COMMA.

Rule 4-Termination: End your plaintext with XX. If necessary, add dummy characters after XX in order to pad out the message to frustrate cryptanalysis and to conclude on a doublet (ensuring the numeric string ends with four digits).

Rule 5: -Use a pointer at the beginning of your message to specify the key so your recipient can decrypt the text.

The Mechanics of Generating Long Keys: What you would like to have is a very long key to use in a scheme like the Vigenère encoding, but as we stated above, the problem with a long key is that you have to transmit that key to the persons who should be able to decode the messages in some secure manner. The more "random" the key is, the more secure it is. To make a really random key, you could imagine using something like a timer connected to a Geiger counter and placing it close to some radioactive source so you could use the intervals between atomic decays to generate numbers that are truly random.

But if you're willing just to have a sequence that looks very random, there are many techniques for generating such sequences of so-called "pseudo-random numbers". Volume 2 of Donald Knuth's classic book, "The Art of Computer Programming-Seminumerical Algorithms"[8], contains a treasure-trove of information about how to generate pseudorandom sequences. First and foremost you may just use the permutation technique explained earlier to randomize your encryption key. In the next section we will look at just a couple of methods here to give you an idea of the possibilities.

A Simple Mechanics of Key Generation: A simple but very easy way to perform a random key setup procedure (but not a very good one) is based on the fact that if p is a prime number and n is any number with 0<n<p then the sequence of numbers: kn (mod p) cycle through all the numbers between 1 and p-1 without missing any of them and in an order that appears somewhat random as k goes through 1,2,3,..., p-1. Modular arithmetic is usually used for this kind of operations. This is called arithmetic in a finite field and makes many common mathematical assumptions untrue (1+1 does not equal two if you are in a finite field of size two)[3]. For real applications we can do the arithmetic over, mod 2^{32}, (2^{32} is round 4 billion). For ease of understanding the concept, let's look at an overly simple example, if p=17 and n=7:

7.1 (mod 17)=7	7.2 (mod 17)=14	7.3 (mod 17)=4	7.4 (mod 17)=11
7.5 (mod 17)=1	7.6 (mod 17)=8	7.7 (mod 17)=15	7.8 (mod 17)=5
7.9 (mod 17)=12	7.10 (mod 17)=2	7.11 (mod 17)=9	7.12 (mod 17)=16
7.13 (mod 17)=6	7.14 (mod 17)=13	7.15 (mod 17)=3	7.16 (mod 17)=10

The numbers cycle through the series: 7, 14, 4, 11, 1, 8, 15, 5, 12, 2, 9, 16, 6, 13, 3, 10. This sequence has the vague appearance of being random. You may further use permutation to achieve a more secure random key. If you choose a larger prime, the sequence will be longer. Using such a sequence, let's encrypt a message: PASSWORD: BIGTIME.

Binary representations work great on a computer, but since many people find base 10 much easier to work with than base 2, the numeric examples developed here will use the encoding shown in Table 2: The entries with XX are not used and "SP" is the space character. Thus, all characters code to between 10 and 99. "W" is 33, "7" is 97 and soon. If we want to encode four characters at a time, for example, we would encode the word "D5TR" as the 8-digit number 14953028. Including the space, the numeric codes for our message (PASSWORD: BIGTIME) letters are: 26, 11, 29, 29, 33, 25, 28, 14, 65, 10, 12, 19, 17, 30, 19, 23, 15

Table 2: Conversion of text to number using decimal representation

	0	1	2	3	4	5	6	7	8	9
0	XX	XX	XX	XX	XX	XX	XX	XX	XX	XX
1	SP	A	B	C	D	E	F	G	H	I
2	J	K	L	M	N	O	P	Q	R	S
3	T	U	V	W	X	Y	Z	a	b	c
4	d	e	f	g	H	I	j	k	l	m
5	n	o	p	q	R	s	t	u	v	w
6	x	y	z	.	,	:	;	‘	“	‘
7	!	@	#	$	%	^	&	*	-	+
8	(	)	[	]	{	}	?	/	<	>
9	0	1	2	3	4	5	6	7	8	9

Now we'll use a combination of the method above to generate a sequence of pseudo-random numbers to use as a sort of Vigenère key. We'll let p=21467 and n=5791. As k goes from 1 to 17 (there are 17 characters in our message), kn (mod 21467) goes through the following 17 numbers: 5791, 11582, 17373, 1697, 7488, 13279, 19070, 3394, 9185, 14976, 20767, 5091, 10882, 16673, 997, 6788, 12579.

We will just encode our numbers by adding these pseudo-random numbers to the numbers in our sequence and taking the result modulo 100. For example, the first letter, 26, is converted to (26+5791)(mod 100)=17. The encoded sequence becomes: 17, 93, 02, 26, 21, 04, 98, 08, 50, 86, 79, 10, 99, 03, 16, 11, 94.

Assuming that the person who wishes to decode the message knows the values of p and n, she can generate the exact same sequence of keys and can undo the encryption above. For example, to decrypt the first character, she generates the first number in the key sequence, 5791, subtracts that from 17 (giving -5774) and taking that result modulo 100 to get 26. Now notice that the only information that has to be passed to the recipient as the key is the pair of numbers p and n-there is no need to transmit a long sequence of keys.

Better Pseudo-Random Sequences: The sequence above is too simple, but with just a tiny modification, it can be made much more interesting. The following sequence can work quite well. Choose a (usually large) prime number p and two integers a and c. In addition, choose a "seed", or starting number for the random sequence which we will call S_0 such that. Here is the formula for S_i for i>0 such that $S_i = (aS_{i-1}+c) \pmod p$.

So, for example, if, p=21467, a=83 and c=651 and we begin with S_0=12121, we generate the following sequence:

$$S_0 = 12121$$
$$S_1 = (83*12121+651) \pmod{21467} = 19212$$
$$S_2 = (83*19212+651) \pmod{21467} = 6689$$
$$S_3 = (83*6689+651) \pmod{21467} = 19163$$
$$S_4 = (83*19163+651) \pmod{21467} = 2622$$
$$S_5 = (83*2622+651) \pmod{21467} = 3607$$
$$S_6 = (83*3607+651) \pmod{21467} = 20961$$

which you can continue as long as you want. The first pseudo-random numbers in this sequence are: 12121, 19212, 6689, 19163, 2622, 3607, 20961. Of course the sequence has to cycle in p or fewer steps. This argument can easily be generalized to keys (and plaintexts) of arbitrary length that should provide unlimited keys for implementing one-time encryption.

Generalized One-Time Pad Implementation: Let's take a look again at our earlier generalized equation for one-time pad encryption, i.e., $C_i=E(P_i, K_i)$. In principle, the encryption E can be achieved by simple look-up in a $256*2^k$-byte table, where k is the size of each key unit. Each column of the table, corresponding to one value of k_i is a permutation of the values 0 to 255. This makes it possible for the receiver to decrypt the message uniquely. The cipher will be most secure if each of these 2^k permutations is statistically independent of the others and of the identity permutation and if the encryption operation is implemented so that the waveforms of the bits of C_i give no clue as to the values of P_i and K_i and, the key must never be reused, ever.

The reason that no portion of the key stream may be reused is that an opponent can detect reuse by the simple Index of Coincidence statistical test. Once the opponent finds two or more portions of messages enciphered with the same part of the key, it becomes possible to decipher those portions. For efficiency purpose, it is good practice to start each message near the position following the key byte used for the last character of the previous message. This eliminates the need to keep track of which portions have been used and removes the danger that a message will be longer than any of the remaining segments.

Key Enhancement Mechanism: Suppose that two parties wish to correspond in secret over potentially compromised channels. They could communicate using a one-time pad, as above, provided that they could obtain a sufficient supply of key stream bytes, or that they could resupply themselves with key bytes at a rate fast enough to prevent ever running out. Note, however, that the total amount of key used over time equals the total amount of message material i.e., one byte of key for each byte of message. So, if they had a secure way of resupplying the key, perhaps they could send their messages this way and skip encryption altogether. Suppose, then, that the parties do not have an endless supply of key bytes, but a limited supply, at least enough for the longest possible message, but perhaps only enough for one day or even one hour of communication.

The question is: therefore, how can their supply be stretched to last a week, or a year, or until new key material can be distributed? There are many ways to

generate a new sequence from a given sequence. Indeed, this is precisely the problem that cryptography addresses. Even such a simple expedient as adding a constant value to each key byte K_i (cf. Caesar cipher) could be effective if the enciphering function E is strong enough[1]. It is safer, though, to choose a method of enhancing the key, which does not depend on either the strength or the secrecy of E. One such method is the concept of autokey.

The Mechanics of Autokey : Autokey is a very simple and fast method of encryption. It has the valuable advantage that each byte of the new key stream has no correlation or fixed relationship with the corresponding byte in the original key stream. Let A and B be two independent simple substitutions. The new key stream R is given by

$$R_1=A(K_1)+B(K_n)$$
$$R_i=A(K_i)+B(R_{i-1}) \text{ for } I=2,3,...,n$$

where, the addition is modulo 256 and n is the length of the old key stream. Greater strength can be obtained by repeating the autokey two or more times in order to "cover its tracks." This assures that in the final iteration none of the original bytes of the K-key stream participate. It is important that the two substitutions A and B are strongly non-linear in the sense that there is no a priori correlation between any subsets of the bits of x and A(x) or B(x). Further, there must be sufficiently many choices of A and B to prevent the correspondents from ever running out of keys. If A and B are determined by permuting the values 0 to 255 using a pseudo-random number generator and this generator has a 40-bit seed, then there are 2^{80} possible R-key streams. Even if each R-stream were used for only one second of communications this supply would last indefinitely. Indeed, if each new key were chosen randomly, rather than systematically, the chance of ever choosing the same R-stream twice is negligible.

Furthermore, since this autokey can be inverted, or reversed, the resulting R-key stream will also be true random. If it were not, then the inversion process would be a deterministic algorithm for producing true-random numbers from nonrandom numbers, which is impossible. By definition, random numbers generated by a deterministic algorithm are pseudo-random. A new key stream may be generated in bulk fashion using the entire K-key stream, or it may be generated as needed using just the portion of the key stream required for the next message. For very long messages, the autokey can be applied to one block of the K-stream at a time as it is retrieved from secondary storage, provided that message keys are not restricted to block boundaries.

Exhaustive Trial Mechanism: Since the K key stream is true random, no method of cryptanalyzing R to recover K would work except exhaustive trial. In order to decide whether 2^{80} possible choices of A and B are adequate for security, any known or potential exhaustive trial attacks should be examined. Assume that an opponent has a copy of the encryption device or software and therefore knows the E operation. Suppose, further, that the opponent knows both the ciphertext and plaintext for some given message. This means that the encryption E can be stripped off, yielding a portion of the R-key stream. It is not feasible for an opponent to try all 2^{80} combinations for the two substitutions and even if that were possible, the K-stream is true random and there is no way to distinguish the correct value.

Further, imagine that the messages sent today will be of value 20 or even 30 years hence, when it might be possible to try 2^{80} keys. The opponent would still need to have two different messages known to be enciphered with the same part of the key stream and compare the 2^{80} possible K-streams for each one. A sample of 12 bytes from each would be sufficient. However, this requires storage on the order of 10^{25} bytes, which seems well beyond what can be projected even 30 years ahead. The autokey, therefore, seems secure against any conceivable attack that could be mounted in the foreseeable future. Of course the key size should be reviewed periodically, as computers get larger and faster. The autokey method allows a relatively small amount of true-random key material to stretch a long, long way.

One warning is necessary: the one-time pad method still depends on the opponent not knowing where in the key stream the key for a specific message begins. The key stream needs to be long enough to make this non-trivial. If P is the peak amount of message material expected during the time period when an L key is in use, L is the maximum length of any message, n is the maximum number of messages expected during the period and G the average gap left between messages in the key stream, then a good rule of thumb would be to make the key stream at least: $P+L+nG+10^6$ bytes long.

The Concept of N-Way Communication: This technique can be extended to N-way communications by using a layered approach. Suppose that there are N-parties (people or stations) that wish to communicate and that all possible pairs of parties may potentially need to exchange messages at some time. It is not sufficiently secure simply to supply all of the participants with the same true-random key stream and let them apply the autokey method above to generate more key material. Two different parties may

use exactly the same key and autokey, thus creating two messages with the same effective key. Such messages will begin appearing when the number of R-streams that have been used reaches about $1.4x2^{40}$. It is considered feasible to record and store $1.4x2^{40}$ messages. An opponent monitoring all pairs of correspondents may detect this and recover part of the effective key, that is, the R-key. This portion might solve messages between these or other pairs.

One solution is to control carefully which keys each pair of correspondents use to generate their R-streams. This poses a huge administrative burden if the number of users is high, especially if users frequently join and leave the system. Worse, if the range of keys allotted to a particular user pair becomes known, then exhaustive trial of that limited set may be feasible, perhaps even easy. A better solution is to use a layered approach. Each pair of correspondents could have a unique autokey used to produce their R-key stream, then they would apply a second autokey to produce the key stream for a specific time period. This way, neither the K-key stream itself nor any derived R-key stream is ever used as the key of a message:

- If the K-stream were discovered by espionage, that would not compromise the communications of any pair. Similarly, no individual with legitimate access to the K-stream could eavesdrop on any of the other parties.
- If the R-stream for a particular pair of parties, or even for several pairs were discovered by espionage, that would not compromise the communications of any other pair, because their R-stream would not be known. Furthermore, it would not compromise the communications of the affected pairs because it would not reveal their key streams for any given time period.

Using this layered approach, a small amount of random key can serve a large number of correspondents over an indefinite period. In fact, there are so many choices for the A and B substitutions that the key can never be exhausted, no matter how many parties join the network.

Summary-About the One-time Pad Security: Provided that an eavesdropper cannot get access to either the sender's or receiver's key, the one-time pad method is 100% secure. No existing National Security Agency's (NSA's) cryptanalyst will ever crack it. No Cray supercomputer running the NSA's cracker software will ever break it. Period. But you need to be prudent about security.

Key security-Good security means you must conceal your key in a location where you'll know if it's been tampered with. Usually this means carrying it on your person at all times.

Location security-Good security means choosing private locations to encrypt and decrypt your messages. Remember, it's easy for your enemies to install a pinhole video camera above your desk. So, when choosing a location, be creative, be unpredictable and, be quick and thorough.

SURVIVAL TIPS-At the first sign of surveillance you must stop working at your desk unless you're absolutely sure there's no way they can gain access to install the video surveillance equipment. In a pinch you can work under your desk until you implement off-site locations.

Disposal security-Good security means destroying your working materials after each encryption or decryption. Don't leave anything around for the enemy to work with. This usually means shredding and burning-or ingestion. (Yes, eat the evidence. It saved the spymaster/espionage kingpin Kim Philby's bacon early in his career).

Random means just that-The security of your one-time pad system depends on the randomness of the key. Don't use a computer to generate your keys. Do it by hand-and be sure to introduce a second element of randomness into your method by throwing dice or flipping a coin every now and then while you're creating your keys.

One-time means just that-Don't use a key more than once. Ever! Even if just a few blocks overlap in two different messages, the NSA cracker software will shift and compare the ciphertext messages until the statistical frequency of characters matches the expected statistics for English language text. Giving the NSA an opening like this is tantamount to setting the fox loose in the hen-house.

The perfect system-When used correctly, the one-time pad system provides perfect security for your secret messages. The weakest link is the human element; therefore, solitary confinement while at work is the best medicine for the ultimate security.

CONCLUSIONS

In this paper we have provided practical information and method that you can use to set up your own one-time pad encryption system. Described also is the subtle refinements that you won't find in other books or articles. For broad application of one-time pad, we have shown that starting with a limited supply of a true-random key, a

simple autokey cipher can be used to generate additional random key material indefinitely. Using a layered approach, the same random key can supply any number of correspondents indefinitely with message keys for use in a one-time pad. Moreover a break in the communications of one or more pairs of correspondents will not compromise the security of any other pairs.

REFERENCES

1. Rabah, K., 2004. Data security and cryptographic techniques-A review. Inform. Technol. J., 3: 106-132.
2. Rabah, K., 2004. Steganography-The art of hiding data. ITJ., 3: 245-269.
3. Rabah, K., 2004. Elliptic curve cryptography. J. Applied Sci., (Submitted).
4. Menezes, A., P. Van Oorschot and S. Vanstone, 1997. Handbook of Applied Cryptography, CRC Press.
5. Kahn, D., 1983. The Codebreakers: The Story of Secret Writing. New York, Macmillan.
6. Ritter and Terry, 1991. The efficient generation of cryptographic confusion sequences. Cryptologia, 15: 81-139.
7. Maurer, U.M. and J.L. Massey, 1990. Perfect Local Randomness in Pseudo-Random Sequences. In: Brassard, G., (Ed.), Advances in Cryptology-CRYPTO'89, LNCS, No. 435, Heidelberg and New York, Springer, pp: 100-112.
8. Knuth, D.E., 1969. The Art of Computer Programming. Vol. 2, Seminumerical Algorithms Addison-Wesley, Reading, Mass.