Attacks on Additive Encryption of Redundant Plaintext and Implications on Internet Security

David A. McGrew and Scott R. Fluhrer

Cisco Systems, Inc.
170 West Tasman Drive, San Jose, CA 95134
{mcgrew,sfluhrer}@cisco.com

Abstract. We present and analyze attacks on additive stream ciphers that rely on linear equations that hold with non-trivial probability in plaintexts that are encrypted using distinct keys. These attacks extend Biham's key collision attack and Hellman's time memory tradeoff attack, and can be applied to any additive stream cipher. We define *linear redundancy* to characterize the vulnerability of a plaintext source to these attacks.

We show that an additive stream cipher with an n-bit key has an effective key size of $n - \min(l, \lg M)$ against the key collision attack, and of $2n/3 + \lg(n/3) + \max(n - l, 0)$ against the time memory tradeoff attack, when the the attacker knows l linear equations over the plaintext and has M ciphertexts encrypted with M distinct unknown secret keys.

Lastly, we analyze the IP, TCP, and UDP protocols and some typical protocol constructs, and show that they contain significant linear redundancy. We conclude with observations on the use of stream ciphers for Internet security.

1 Introduction

Biham's key collision (KC) attack [5] and Hellman's time-memory tradeoff (TMTO) attack [11] can be adapted to attack additive encryption in the case that many ciphertexts encrypted with distinct keys, whose corresponding plaintexts all obey some known linear relations, are available to the cryptanalyst. Both of these methods use a precomputation stage in which some knowledge of the typical plaintext is used to build a database, followed by an attack stage in which (hopefully many) ciphertexts are analyzed in an attempt to find unknown keys. The computational cost of the precomputation stage can be amortized over many runs of the attack stage, significantly reducing the effective key size of the cipher against these attacks. These attacks rely on the fact that there are linear equations in the plaintext bits that are known to the cryptanalyst. We define the *linear redundancy* of a plaintext source as the to capture this property.

A linearly redundant source may involve linear equations that hold with probabilities that are not close to unity. We present and analyze an adaptation of the KC attack that works in such cases by using error correcting codes.

D.R. Stinson and S. Tavares (Eds.): SAC 2000, LNCS 2012, pp. 14–28, 2001.

We analyze the linear redundancy of the Internet Protocol (IP), the Transmission Control Protocol (TCP), and the User Data Protocol (UDP) traffic with stream ciphers. IP is used by the Internet to transport packets between networks [1], while TCP and UDP are the most common higher-level protocols transported by IP. IP, TCP, and UDP packets are known to contain a significant amount of data that is guessable by an adversary (as was pointed out by Bellovin [4]). Our analysis extends these observations by showing that these packets contain a large amount of linear redundancy that can be used in cryptanalytic attacks.

The Stream Cipher ESP (SC-ESP) is a specification for the use of those ciphers to provide privacy within the IPSEC framework [13,9]. It describes how to use additive stream ciphers for the encryption of IP packets (if used in tunnel mode) as well as TCP, UDP, or other packets (if transport mode)[1]. Below, we derive requirements on SC-ESP that provide protection against the attacks that we develop in this paper. We do not investigate the linear redundancy of other important Internet protocols, such as HTTP or RTP, though such protocols are commonly used with additive encryption in the SSL, TLS, and SSH protocols. However, the techniques that we develop in this paper do apply to their analysis, and we expect that these protocols also contain a significant amount of linear redundancy.

The rest of this paper is organized as follows. Section 1.1 introduces our terminology and assumptions. Section 2 introduces the idea of linear redundancy. Section 3 introduces the key collision attack, shows how it can be applied to attack additive encryption, and analyzes its computational cost and success probability, while Section 3.1 shows how that attack can be modified to deal with linear equations that are probabilistic, rather than deterministic. Section 4 adapts Hellman's time-memory tradeoff to attack additive encryption, and analyzes the resulting algorithm. The IP, TCP, and UDP protocols are analyzed in Section 5, and are shown to contain enough linear redundancy to enable the successful prosecution of the attacks that we derive. Our conclusions are presented in Section 6.

1.1 Terminology and Assumptions

An additive stream cipher is a cipher that encrypts a plaintext by bitwise adding it (modulo two) to a keystream. The keystream is generated pseudorandomly, given a secret key. Mathematically,

$$c_i = p_i \oplus s_i(k), \tag{1}$$

where c_i, p_i and $s_i(k)$ are the i^{th} bit of the ciphertext, plaintext, and the keystream corresponding to the key k. Additive stream ciphers can be defined over any group, and our results can easily be generalized, but below we consider only binary addition for clarity of exposition.

Modern stream ciphers include RC4, SEAL, the Output Feedback (OFB) mode specified by NIST for use with the DES [18] and the counter mode for

[1] See [8,9] for a more detailed description of IPSEC and ESP

block ciphers [16, p.100]. RC4 is widely used to secure HTTP, as it is part of the Secure Sockets Layer (SSL) and Transport Layer Security (TLS) specifications. Other stream ciphers in use include the recently broken A5/1 [2] used in GSM cellular phones, and the cipher E0 in the Bluetooth specification for Wireless LAN Security [3].

We make the conventional assumption that the cryptanalyst can check if a key is correct by trial decryption of a ciphertext followed by a redundancy check of the decrypted plaintext. We also assume that ciphertexts are distributed uniformly at random, which is essentially equivalent to assuming that the cipher is indistinguishable from a truly random source. We also assume that the cryptanalyst has access to many ciphertexts encrypted under many distinct keys, whose corresponding plaintexts originate from a random but redundant source whose mathematical characterization is known to the attacker. We make the implicit assumption that the unknown keys are distinct, which is a good assumption when the number of unknown keys is less than the square root of the total number of keys, from the 'birthday paradox'.

2 Linear Redundancy

We generalize the idea of known or guessable plaintext attacks by considering attacks on a large number of ciphertexts whose plaintexts were all generated by the same source. We use the information theoretic idea of a plaintext source as a generator of binary strings that chooses strings by a random process that can be characterized by a probability distribution. A source is *redundant* when its probability distribution is not uniform.

To attack an additive cipher, we consider linear equations in terms of the plaintext bits. From Equation 1, it follows that

$$c_i \oplus c_j = (p_i \oplus p_j) \oplus (s_i(k) \oplus s_j(k)). \tag{2}$$

If $p_i \oplus p_j$ is zero (respectively, one), then $c_i \oplus c_j$ will equal $s_i(k) \oplus s_j(k)$ (respectively, will be its opposite). If the same property holds for a large number of plaintext bits, those bits can be used to identify a collision between a secret key set and a known key set. A single linear relation among the plaintext bits of all plaintexts from a source is equivalent to a single bit of known plaintext, for our purposes.

If there are l linear relationships between the plaintext bits, this fact can be represented mathematically as

$$\bigoplus_{i=1,}^{w} L_{ij} p_i = e_j \text{ for all } j : 1 \leq j \leq l, \tag{3}$$

where L_{ij} is an invertible $m \times w$ boolean matrix, and e is an $l \times 1$ boolean vector.

More generally, Equation 3 can hold with some probability not equal to one. The vector $\delta = Lp \oplus e$, which would be the zero vector if the linear equations held with probability one, has a low hamming weight. We define the set Δ as the set of typical (e.g., most probable) values of δ, such that

$$\sum_{\delta \in \Delta} P(\delta) = 1 - \epsilon, \tag{4}$$

where $P(\delta)$ is the probability that $Lp \oplus e$ will have the value δ, and ϵ is a number less than one. We say that a plaintext source has *linear redundancy* (λ, ϵ) if there exists an L and e as defined above such that $\lambda = 1 - \lg(\#\Delta)/l$.

In the case that each of the linear equations hold with the same probability ϕ, then the expected weight of $\delta = Lp \oplus e$ is ϕl. In this case, the size of the set of typical vectors Δ is well approximated by $\sum_{i=0}^{\phi l} \binom{l}{i} \simeq 2^{lh(\phi)}$, where $h(\phi) = -\phi \lg \phi - (1-\phi) \lg(1-\phi)$ is the binary entropy function[2]. The linear redundancy then reduces to $(1-h(\phi), 1/2)$. In the following, we focus on the practical attacks rather than the theoretical characterization of linear redundancy.

Our attacks can be viewed as decoding unknown keys, and thus matching them to some set of known keys. From this viewpoint, there is a noisy communication channel from an unknown key to the attacker, where the 'noise' is a plaintext message. The unknown keys are the source words, the keystream segments are the code words, ciphertexts are the received words. The attacker faces the problem of decoding the received words to a known code word. We call this channel the *cryptanalytic channel*, and it is analogous to the one defined by Siegenthaler in the description of correlation attacks on combination generators [14]. The code used in our attacks is a set of keys that is randomly chosen by the attacker. Obviously, ciphertexts created with unknown keys that are not in the code cannot be properly decoded. Our attacks work by decoding correctly whenever possible, and rely on the 'birthday paradox' to ensure that there are keys common to both the random code and the set of unknown keys.

In some cases, attacks using linear redundancy can be significantly improved through the use of traffic analysis, that is, the use of external information about the ciphertexts to establish the value of the vector e. In the case of Internet security, this information includes the length of the encrypted data, the time of creation of the encrypted data, and the position of each ciphertext in the sequence of all ciphertexts.

3 Key Collision Attacks on Additive Encryption

Key collision attacks [5] take advantage of the birthday paradox to reduce the expected work effort of finding secret keys. These attacks use two distinct sets of keys: a set of unknown secret keys, and a set of keys generated by the cryptanalyst. These sets will contain a common element with high probability when the product of the sizes of the sets is close to the size of the set of all keys.

The known-plaintext key collision attack works as follows: the cryptanalyst encrypts the same fixed plaintext with N distinct keys, and stores the resulting

[2] This approximation uses the tail inequality [6], and is asymptotically exact

ciphertexts along with the keys that generated them. We call the set of ciphertexts the *known key* set. The cryptanalyst then gets a hold of M ciphertexts that are created by encrypting the same plaintext with distinct unknown keys, and looks for collisions, that is, elements with the same keys that are in both sets. When one of the unknown keys is equal to one of the known keys, a collision occurs. With an n-bit key, this will happen with high probability when $MN \geq 2^n$. In practice, many keys may map the same plaintext to the same ciphertext, so the cryptanalyst must check each collision with a trial decryption.

In order to attack additive encryption of linearly redundant plaintext, we define a *hallmark* of a key. This is a binary vector that captures enough information about the key to enable elements of the known key set to be matched to the unknown key set.

Combining Equations 3 and 1 gives

$$\bigoplus_{i=1,}^{w} L_{ij} s_i(k) = e_j \oplus \bigoplus_{i=1,}^{w} L_{ij} c_i \text{ for all } j : 1 \leq j \leq l. \tag{5}$$

The *known key hallmark* v is defined by $v_j(k) = \bigoplus_{i=1,}^{w} L_{ij} s_i(k)$. The *unknown key hallmark* u is defined by $u_j = e_j \oplus \bigoplus_{i=1,}^{w} L_{ij} c_i$. Both v and u are length l binary vectors. In the event that $u = v$, it is (at least relatively) likely that the known key and unknown key are equal.

We now show how to prosecute a KC attack on an additive cipher, given L and e. In the precomputation stage, compute the set $V = \{(v(k), k) : k \in R\}$ of known keys and their hallmarks , where R is a set of N arbitrary distinct keys, and sort the vectors so that their first components are in non-decreasing order. In the attack stage, we are given the set $C = \{c\}$ of ciphertexts, and we want to find as many of the unknown keys as possible. We denote the number of ciphertexts (and thus the number of unknown hallmarks) as M. The attack algorithm follows:

1. Compute the set of unknown keys and hallmarks $U = \{(Lc \oplus e, c) : c \in C\}$, and sort it into non-decreasing order.
2. Find the join $J = \{(x, k, c) : (x, c) \in U, (x, k) \in V\}$, that is, the intersection of the first components of V and U.
3. For each element $(x, k, c) \in J$, do a trial decryption of the ciphertext c using the key k.

The intersection of two sets of bit vectors can be found by sorting each set into non-decreasing order, maintaining a pointer into each set, repeatedly advancing the pointer that points to the smallest element, and outputting the elements when they match [10]. If radix sort [10] is used, then this algorithm is completely paralellizable.

An important property of this attack is that the vector e need not be known during the precomputation stage; it is sufficient to know e during the attack stage. This property lends itself to practical attacks, as there are many cases in which the plaintext at two locations will be linearly related, though the exact value of the relationship may only be predictable through traffic analysis or other external means. For example, traffic analysis of the IP protocol can readily discern many TCP/IP packets (TCP ACK packets have the distinctive length of 43 bytes), thus revealing the value of the 'Protocol' field of the IP packet (See Table 1).

The basic key collision attack requires storage of order $M + N$. The precomputation stage requires N encryptions and $N \lg N$ comparisons and copies (for sorting).

The attack stage requires $M \lg M$ comparisons and copies in the sorting stage. Finding the set J requires $M + N$ comparisons. The attack performs $\#J$ trial decryptions, which is equal to the sum of the number of false hits, which is $MN/2^l$, and the number of true hits, which is $MN/2^n$. The total computation is of of order $M \lg M + M + N + MN(1/2^l + 1/2^n)$.

The expected number of true hits found, that is, the number of messages successfully decrypted, is $MN/2^n$. Thus, the order of the expected work w for each successful decryption is given by

$$\begin{aligned} w &= \frac{M \lg M + M + N + MN(1/2^l + 1/2^n)}{MN/2^n} \qquad (6) \\ &= \frac{2^n \lg M}{N} + \frac{2^n}{M} + \frac{2^n}{N} + 2^{n-l} + 1. \end{aligned}$$

If $2^n \lg M/N$ is the leading term in Equation (6), we say that the attack is *sort limited*. If $2^n/M$ is the leading term, we say that the attack is *intersection limited*. This can happen when the size of the known key set is large and the size of the unknown key set is small. If 2^{n-l} is the leading term, we say that the attack is *information limited*. This case happens when there are few linear equations in the plaintext. The term 1 can never be the leading term, as this implies that the known and unknown key sets are larger than the set of all keys. This term can be neglected in practice. The expected number of keys that are tried for a given unknown key hallmark is $N/2^l$. When the attack is information limited, this number is large (on average). When then attack is sort limited or intersection limited, it is small (on average).

To make the advantage over exhaustive search explicit, we introduce the *effective key size*, which we define to be the base-two logarithm of the order of the expected work. The effective key size is denoted as η, and is given by

$$\eta = \lg w = n + \lg \left[\frac{1 + \lg M}{N} + \frac{1}{M} + 2^{-l} + 2^{-n} \right]. \qquad (7)$$

When the attack is information limited, then

$$\eta \simeq n + \lg 2^{-l} = n - l. \qquad (8)$$

This approximation is valid when $l \ll \lg \min M, N$. The linear relationship between effective key size and l is shown in Figure 1. When the attack is intersection limited, then

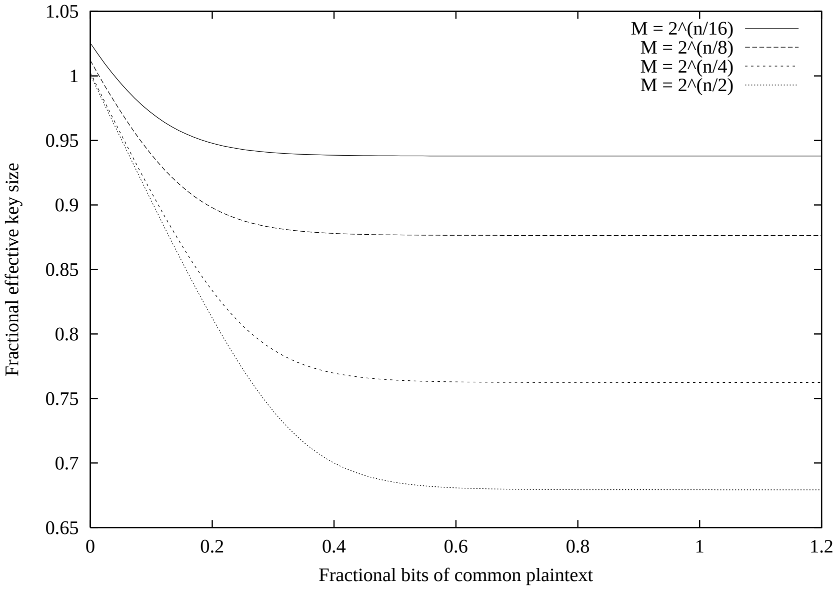

Fig. 1. The effective key size as a function of the linear redundancy. In this figure, the fractional key size η/n is plotted versus the fractional number of linear equations l/n. The plot shows various values of $M/2^n$; in every case, $N = 2^{n/2}$

$$\eta \simeq n + \lg\left(\frac{1}{M} + \frac{1}{N}\right) \simeq n - \lg \min M, N. \tag{9}$$

When the attack is sort limited, then

$$\eta \simeq n + \lg \frac{\lg M}{N} = n - \lg N + \lg \lg M. \tag{10}$$

We define the *break even value* N_b to be the value of N such that the effective key size is equal to the actual key size; when N is below this value, the attack is not effective. Solving for this value, we find that $N_b = (1 + \lg M)/(1 - 2^{-l} - 2^{-n} - 1/M)$. The effective key size as a function of N can be succinctly expressed as

$$\eta = n - \lg\left[1 + (1 + \lg M)\left(\frac{1}{N} - \frac{1}{N_b}\right)\right]. \tag{11}$$

The dependence of η on N is demonstrated in Figure 2. The maximum possible value for N_b is 2^n, which implies that a necessary condition for the above attack to provide an advantage over exhaustive search is that

$$l \geq \lg\left(1 - \frac{1}{M} - \frac{1 + \lg M}{2^n}\right). \tag{12}$$

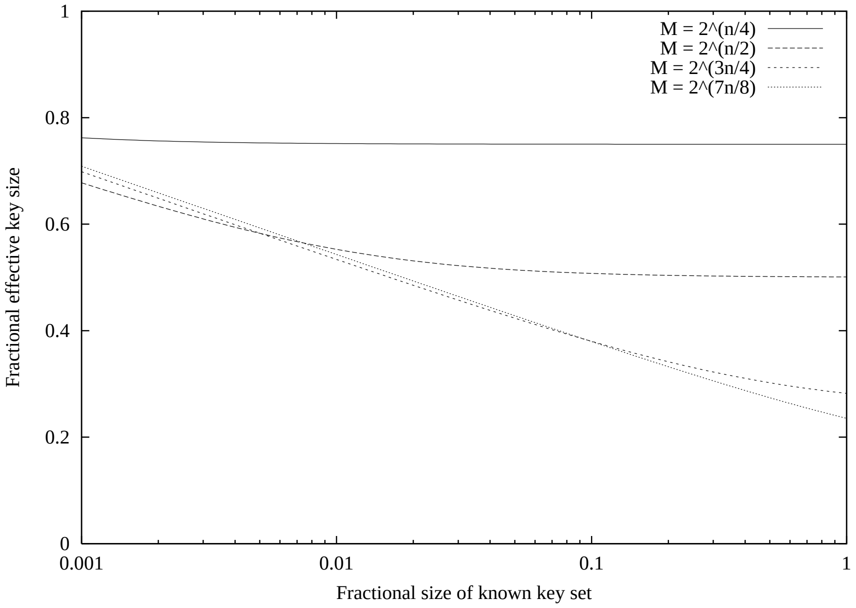

Fig. 2. The effective key size as a function of the size of the unknown key set. In this figure, the fractional key size η/n is plotted versus the fractional size of the known key set $N/2^n$. In these plots, $m = n$

3.1 Probabilistic Linear Equations

The KC attack on additive encryption in the previous section can work even when the linear equations (5) do not always hold, but hold with some non-negligible probability. If the probability that all of the equations hold simultaneously is p, then the effective key size is increased by $\lg 1/p$. However, better attacks can be realized by using error-correcting codes. Below we present a simple adaptation of the KC attack that uses error correction of the hallmarks.

The error-correcting KC attack differs from the KC attack presented above in the precomputation stage and in Step 1. The known key hallmarks are required to be codewords of an error-correcting code D which has codewords of length l, a total of 2^k codewords, and which can correct up to e errors. This property can be realized by using a rejection method during the precomputation stage, which will increase the amount of computation in that stage by a factor of about 2^{l-k}.

Step 1 of the attack algorithm is modified by changing the definition of the set U to $U = \{(d(Lc \oplus e), c) : c \in C\}$, where d is a decoding function for the code D.

The effective key size can be derived as with the information limited case above, with the differences that in the error correcting case the number of false hits is now $MN/2^k$ and the number of true hits is $pMN/2^n$ [3]. The effective key size η_{EC} for the error-correcting KC attack is

$$\eta_{KC} \simeq n - k + \lg \frac{1}{p} = n - lR + \lg \frac{1}{p}, \tag{13}$$

where $R = k/l$ is the rate of the code. There is a tension between R and the decoding error, in that increasing one tends to decrease the other. It is difficult to further characterize the effectiveness of this attack in the general case because of the variety and complexity of error correcting codes [12]. One example of a useful code is the $n = 128, k = 100, e = 4$ code based on BCH codes [15]. Gallager codes [7], which have proved useful in correlation attacks on stream ciphers, may also prove useful in our attacks. It is also possible to use nonlinear codes, though such codes could require a significant storage space.

A strict lower bound on the effective key size of the error correcting KC attack is provided by an information theoretic treatment of the cryptanalytic channel. The capacity C of that channel, which is determined by the plaintext source [6], is the upper bound on the rate R of a code that can be used in the attack, thus limiting that value in Equation (13).

4 Hellman's Time-Memory Tradeoff

Hellman's time-memory tradeoff (TMTO) is a method that can be used to dramatically reduce the average amount of computation needed to invert a one-way function [11]. It works by precomputing a large table, then using the same table to attack many secret keys. Asymptotically, this attack can be used to break a block cipher with an n bit key with about $2^{2n/3}$ operations, using $2^{2n/3}$ storage [11]. Below, we review how to invert a function $S : \mathbb{F}_2^n \rightarrow \mathbb{F}_2^l$, then show how to adapt this result to attack additive encryption of linearly redundant plaintext.

To perform the TMTO, given the function S to be inverted, select a reduction function $R : \mathbb{F}_2^l \rightarrow \mathbb{F}_2^n$, the size of the table N[4], and the tradeoff parameter t. The reduction function serves to map the range of S back onto its domain. In the precomputation stage, define the function $f(x) = R(S(x))$, and compute the set $T = \{(f^t(x), x) : x \in R\}$, where R is a random N-element subset of $\mathbb{F}_2^n$, and sort the elements of T so that their first components are in increasing order.

In the attack stage, to find z given y such that $S(z) = y$, compute the set $Y = \{(f^i(R(y)), i) : i = 0, 1, \dots , t - 1\}$, and sort its elements so that their first components are in increasing order. For each component $(a, i) \in Y$ such that $(a, x) \in T$ for some x, compute $f^{t-i-1}(x)$ and check if it is the proper inverse.

The precomputation stage requires Nt evaluations of the function f, as well as $N \lg N$ operations for the sorting and storing N elements. The inversion stage

[3] This analysis assumes that the decoding function is equally likely to chose any codeword if the number of errors in the hallmark is greater than e, a property which holds for linear codes

[4] Hellman refers to this parameter as m in [11]. We use the notation N to be consistent with the terminology in the Key Collision section

requires t evaluations of f, sorting and storing t elements, and $N+t$ operations to find the intersection of a t element set and an N element set.

In the TMTO attack against block ciphers, the work effort due to false hits is negligible. R can be chosen so that it does not collide, so that a collision of S implies a collision of the the underlying function f. If $l < n$, then this work effort is no longer negligible, as the function f will have more collisions than expected. The expected number of false hits per table look up is bounded by $Nt(t+1)/2^{l+1} \simeq Nt^2/2^{l+1}$.

The success probability of the TMTO attack algorithm is determined by the number σ of elements in the known key set V, where σ can be bounded by,

$$\sigma \leq \sum_{i=1,}^{N} \sum_{j=1,}^{t} \left[\frac{2^n - it}{2^n}\right]^j . \tag{14}$$

Using the choice of parameters $N = t = 2^{n/3}$ suggested in [11], then $\sigma \simeq 2^{2n/3}$. Below, we assume these values.

To use Hellman's time-memory tradeoff in an attack against linearly redundant plaintext encrypted with a stream cipher, f is defined as a mapping from keys to known hallmarks :

$$f(k) = Ls(k) \oplus e. \tag{15}$$

The known key set of hallmarks is the 'logical table' comprised of the iterates of S used in computing T, and the number of distinct elements that it contains is σ.

If the TMTO is done on a set of M unknown hallmarks simultaneously, a set Y must be computed for each unknown hallmark, and the union of the unknown key sets has cardinality tM. In addition, if $n > l$, the time taken to check false hits must be accounted for. The expected work effort w of the TMTO attack is thus for $n \leq l$,

$$\begin{aligned} w &= tM\lg(tM) + tM + N + \frac{tM\sigma}{2^n} \qquad (16)\\ &= O(tM\lg(tM) + N). \end{aligned}$$

and for $n > l$,

$$\begin{aligned} w &= tM\lg(tM) + tM + N + \frac{tM\sigma}{2^n} + MNt^3/2^{l+1} \qquad (17)\\ &= O(tM\lg(tM) + N + MNt^3/2^l). \end{aligned}$$

The expected number of correct keys that this algorithm finds is $M\sigma/2^n \approx M2^{-n/3}$. Thus the effective key size η_T of the TMTO attack is given by

$$\begin{aligned} \eta_T &= \lg\frac{tM\lg(tM) + N + MNt^3/2^l}{M2^{-n/3}} \qquad (18)\\ &= 2n/3 + \lg\left(n/3 + \lg M + 1/M + 2^{n-l}\right)\\ &\simeq 2n/3 + \lg n/3 + \max(n-l, 0). \end{aligned}$$

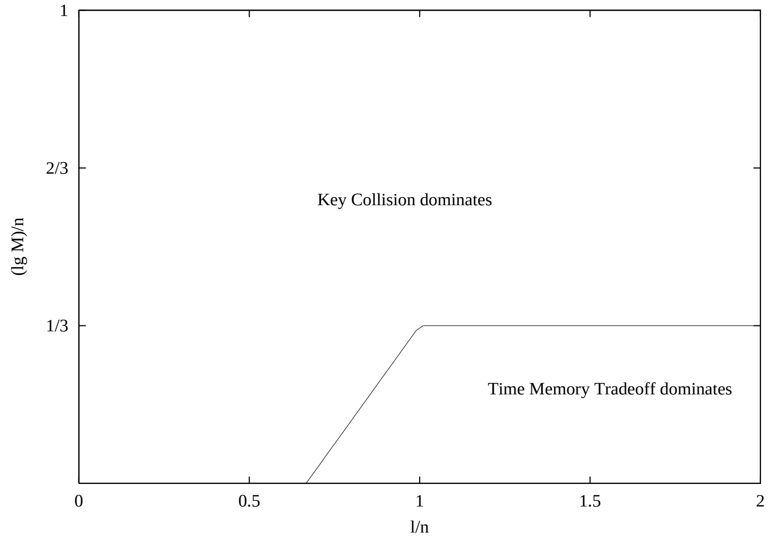

Fig. 3. The ‘phase space’ of attacks on additive encryption, showing which attack dominates as a function of the parameters lg M and l. Here, the parameters are represented fractionally in terms of the key size n

4.1 Comparison of the TMTO and Key Collision Algorithms

The TMTO attack is more effective than the basic attack when $\eta_T < \eta$. By comparing the estimates for η and η_T given above, we can see that the KC attack is preferable when $\lg(M) > n/3$. The complete ‘phase space’ of attacks on additive encryption is illustrated in Figure 3.

However, the TMTO as described does not work for probabilistic linear equations. In that case, the KC attack has the advantage.

5 Linear Redundancy in IP Packets

We analyzed the IP, TCP, and UDP protocols, and estimated the linear redundancy in the headers of those protocols. A summary of our results is given in Table 1. In this section, all numerals indicate binary expressions.

The Version field is (almost without exception) equal to 0100. The Header Length is nearly always equal to 0101, unless an IP option is used, in which case it is probably 0110. The Precedence/TOS (Type of Service) field is generally set to 00000000. The Protocol field is usually 00000110 (for TCP) or 00001011 (for UDP). The ‘Time to Live’ field is usually 00010000 or less. The ‘Source IP’ and ‘Destination IP’ fields from the IP header are the same in every packet between to particular hosts. Each pair of packets with the same source and destination that can be identified by traffic analysis provides 64 linear equations. The ‘Source Port’ and ‘Destination Port’, in the TCP and UDP protocols, provide a total of

Table 1. Linear redundancy in the headers of the IP, TCP, and UDP protocols. The common values are described in Section 5. The 'Single Packet' column shows the redundancy that is detectable in a single packet. The 'Two Packet' column shows the redundancy that is present in two consecutive packets from the same source

Protocol	Field	Size (bits)	Single Packet Redundancy (bits)	Two Packet Redundancy (bits)
IP	Version	4	4	4
	Header Length	4	4	4
	Precedence/TOS	8	8	8
	Packet Length	16	4	4
	Packet ID	16	0	0
	DF bit	1	1	1
	MF bit	1	0	0
	Fragment Offset	13	0	0
	Time to Live	8	3	3
	Protocol	8	7	7
	Checksum	16	1	1
	Source Address	32	0	32
	Destination Address	32	0	32
	Total	-	32	96
UDP	Source Port	16	0	16
	Destination Port	16	0	16
	Length	16	0	0
	Checksum	16	1	1
	Total	-	1	33
TCP	Source Port	16	0	16
	Destination Port	16	0	16
	Sequence Number	32	0	18
	Ack. Number	32	0	14
	Data Offset	4	4	4
	Checksum	16	1	1
	Urgent	8	0	0
	Total	-	5	69

32 linear equations in the same manner. The TCP 'Data Offset' field is usually set to 0101.

5.1 Checksums and Counters

Many protocols use checksums so that transmission errors are likely to be detectable by the receiver. A checksum is an element of a ring, usually $\mathbb{F}_2^c$ or $\mathbb{Z}/2^c$, for some value of c. It is computed by decatenating the data into elements of that ring, then summing them together. Checksums over $\mathbb{F}_2^c$ are conventional when the protocol is implemented in hardware, while checksums over $\mathbb{Z}/2^c$ are commonly implemented in software (and are used for IP, TCP, and UDP with $c = 16$.).

A checksum over $\mathbb{F}_2^c$ (or CRC) provides c linear equations that always hold. The Bluetooth specification for wireless networking is one example of a protocol that includes such a checksum on data that is encrypted by an additive cipher [3]. A checksum over $\mathbb{Z}/2^n$ provides one linear equation that always holds, since the least significant bit of a sum of integers is equal to the exclusive or of the least significant bits of the integers. Probabilistic linear equations in other bits of the checksum can be derived, but will be poor approximations if the number of integers summed together is large.

In many protocols, an integer called a *counter* is included in each packet, and is used to indicate the ordering of the packets to the receiver. Counters may be incremented by one for each new packet, or may be incremented by some other value (e.g., the number of bytes contained in the data portion of the packet, as is done in the TCP protocol). If a c-bit counter x appears in a packet, and $x + y$ appears in another packet, where $y < 2^q$, for some q, then

$$x_{i+q} = (x + y)_{i+q} \text{ with probability} \geq 1 - 2^{-i}. \tag{19}$$

A c bit counter that increments by a value less than 2^c provides a significant amount of information.

IP, TCP, and UDP all use checksums over $\mathbb{Z}/2^{16}$. The low bit of the checksum is a linear function of the other packet data, from Section 5.1. If the layer three protocol of a packet is known, then the checksums provide two linear equations that hold with probability one.

TCP packets contain a 32-bit counter that is incremented by the length (in bytes) of the packet's data. These lengths will be no more than 1500 (which is the Ethernet MTU) with high probability. Since $2^{11} > 1500$, two adjacent counters provide 18 linear equations that hold with probability 7/8 or greater. To use these equations in an attack requires some traffic analysis to discover two sequential TCP packets. The TCP 'Acknowledgement Number' similarly provides about 14 linear equations.

6 Conclusions

Practical attacks on additive stream ciphers that rely on linear equations over the plaintext bits are possible, even when those equations hold probabilistically. The IP, TCP, and UDP protocol headers have a significant amount of linear redundancy, and are vulnerable to these attacks. In practice, effective key sizes of Internet encryption are close to $n - \lg M$, when a cryptanalyst has M ciphertexts encrypted under distinct keys available. We conjecture that only protocols specifically designed to not be linearly redundant will not be vulnerable to these attacks. Compression would reduce the linear redundancy of a source; however, we are pessimistic about the effectiveness of using compression to protect against our attacks in practice.

While our attacks are powerful, there is an easy defense against them: increase the key size of the cipher. Cipher keys can be extended in ways that are not secure against other forms of attack (e.g., 'whitening' with a fixed value) and still provide resistance to our attacks. This approach is similar to the idea of concatenating 'salt' (e.g., unique but public data) to a secret password in order to reduce the effectiveness of attacks that amortize effort across many passwords. variable size key, although in common usage its key size is 128 bits.

The attacks that we outline are possible against Internet traffic encrypted with 128-bit RC4 with a complexity of about 2^{88}, assuming that an adversary can intercept ciphertexts from 2^{40} distinct sessions. This number is feasible; a single Internet site that establishes 2^{32} SSL connections per day has been reported [17]. While this attack is beyond the limit of current cryptanalytic technology, it is worth noting that it does no harm to increase the key size to compensate for our attacks: the throughput of the RC4 cipher is independent of its key size.

The attacks that we presented rely on the fact that the secret keys are chosen uniformly at random. If the keys are chosen from a highly skewed probability distribution (e.g., a broken random number generator that outputs the same number every time), the effectiveness of our attacks is significantly reduced. Of course, the broken random number generator creates other security problems!

Considerable future work remains untouched. While we established the viability of attacks relying on the redundancy of plaintext encrypted by additive stream ciphers, we did not investigate efficient decoding methods for use when the linear equations are probabilistic. Also, it may be possible to extend the time-memory tradeoff approach so that it can be used in the probabilistic case.

References

1. Postel, J., "The Internet Protocol", IETF RFC 791, USC/Information Sciences Institute, September 1981.
2. Briceno, M., Goldberg, I., Wagner, D., "A pedagogical implementation of A5/1", http://www.scard.org, May 1999.
3. Bluetooth SIG, "BLUETOOTH Baseband Specification Version 1.0 B, Section 14.3", http://www.bluetooth.com.
4. Bellovin, S., "Probable Plaintext Cryptanalysis of the IP Security Protocols," Proceedings of the Symposium on Network and Distributed System Security, pp. 155-160, 1997.
5. Biham, E., "How to Forge DES-Encrypted Messages in 2^{28} Steps", Technion Computer Science Department Technical Report CS0884, 1996.
6. Blahut, R., "Principles and Practice of Information Theory", Addison-Wesley, 1983.
7. Gallager, R., "Low Density Parity Check Codes", IEEE Transactions on Information Theory, IT-8 pp. 21-28, January, 1962.
8. Kent, S., and R. Atkinson, "Security Architecture for IP", RFC 2401, November 1998.
9. Kent, S., and R. Atkinson, "IP Encapsulating Security Payload (ESP)", RFC 2406, November 1998.
10. Knuth, D., "The Art of Computer Programming: Volume Three, Sorting and Searching", Addison-Wesley, 1998.
11. Hellman, M. E., "A cryptanalytic time-memory trade-off", IEEE Transactions on Information Theory, July 1980, pp. 401-406.

12. Litsyn, S., Rains, E.M., Sloane, N.J., "Table of Nonlinear Binary Codes", http://www.research.att.com/ njas/codes/And.
13. McGrew, D., Fluhrer, S., "The Stream Cipher Encapsulating Security Payload," draft-mcgrew-ipsec-scesp-01.txt, Internet Draft, July, 2000.
14. Siegenthaler, T., "Correlation-immunity of nonlinear combining functions for cryptographic applications," IEEE Transactions on Information Theory, Vol. IT-30, pp. 776-780, October, 1984.
15. Sloane, N. J, Reddy, S., and Chen., C., "New binary codes". IEEE Transactions on Information Theory, IT-18, pp. 503-510, 1972.
16. Schneier, B., "Applied Cryptography", New York: Wiley, 1996.
17. van Someren, N., "There will be no cryptographic abundance without cryptographic hardware", Xerox PARC Symposium on Life in an Era of Cryptographic Abundance, June, 2000.
18. U. S. National Institute of Standards and Technology, "DES Modes of Operation", Federal Information Processing Standards Publication 81, 1980.