A Space/Time Trade-Off in Exhaustive Search Attacks on Stream Ciphers

Steve Babbage[1]
Vodafone Ltd, Newbury, UK

9 April 1996

Abstract: We present a simple technique which can speed up exhaustive search attacks on stream ciphers, at the expense of requiring more known keystream and more computer memory. The technique is very general in its application. From it we suggest a design principle for stream ciphers.

1. Binary Stream Ciphers

1.1 Model of a Binary Stream Cipher

We model a binary keystream generator as shown in Figure 1. The right arrow represents the state transition function, and the downward arrow represents the output function which computes the tth keystream bit from the tth KG state. The key selects the initial state S_0.

Let $\boldsymbol{S}$ be the set of all possible KG states, and let $n = \log_2(|\boldsymbol{S}|)$, so that the number of possible KG states is equal to 2^n.

$$
\begin{array}{ccccccccc}
S_0 & \rightarrow & S_1 & \rightarrow & \cdots & \rightarrow & S_t & \rightarrow & \cdots \\
\downarrow & & \downarrow & & & & \downarrow & & \\
k_0 & & k_1 & & \cdots & & k_t & & \cdots
\end{array}
$$

Figure 1: The sequences of states and keystream bits in a binary keystream generator

1.2 Exhaustive Search

The standard exhaustive search attack assumes knowledge of at least n bits of information about the keystream, and involves a search of all possible KG states, which will of course require time of order at least 2^n. Very little computer storage is needed.

1.3 The Principle of the Improved Attack

Suppose we have obtained a number of consecutive known keystream bits. Let the number of known keystream bits be 2^m+n-1, so that we can extract a list of $M=2^m$ overlapping n-bit subsequences.

Now suppose that we generate $R=2^r$ independent random KG states, and for each such state produce the n-bit keystream subsequence obtained by starting the KG in that state.

---

[1] Work carried out while at Racal Comsec Ltd, Salisbury, UK

We now have a list of 2^m n-bit subsequences of known keystream, and a list of 2^r n-bit subsequences of keystream generated from known KG states. If a subsequence on the first list matches a subsequence on the second, then with high probability the known state which generated the subsequence on the second list is the same as the state of the KG at the appropriate point in the generation of the known keystream. For such a match to be likely we need $m+r \approx n$.

The point of the attack is that the process of searching for a match between the two lists can take far fewer than 2^n operations. We learn the KG state at some point in the generation of the known keystream; this certainly allows us to generate all subsequent keystream, and in most real life systems will also allow us to work back to the initial KG state.

1.4 First Formulation of the Attack

Suppose we know the keystream bits $k_{s+1} \dots k_{s+M+n-1}$. We list the $M=2^m$ overlapping n-bit subsequences:

$$2^m \text{ subsequences} \begin{cases} k_{s+1} & \cdots & k_{s+n} \\ k_{s+2} & \cdots & k_{s+n+1} \\ \vdots & \ddots & \vdots \\ k_{s+M} & \cdots & k_{s+M+n-1} \end{cases}$$

We then sort this list of subsequences (e.g. into lexicographic order); this requires storage of order $(n+m)2^m$ and time of order $(n+m^2)2^m$.

Now repeatedly select a random KG state S, generate an n-bit keystream sequence by starting the KG in state S, and check the list for an exact match. It is easy to check for the occurrence of a single item in a sorted list; it will require time of order m^2. We expect to need roughly 2^{n-m} attempts before a match is found.

Choosing $m=n/2$, the entire attack will require a number of known keystream bits of order $2^{n/2}=\sqrt{|\mathbf{S}|}$, time of order $n^2 2^{n/2} \approx \sqrt{|\mathbf{S}|}$, and space of order $n 2^{n/2} \approx \sqrt{|\mathbf{S}|}$.

1.5 Second Formulation of the Attack

In the second formulation of the attack we perform the sorting in a one-off pre-processing phase, the results of which can be used to attack several different instances of use of the KG.

In the pre-processing phase we choose $R=2^r$ random KG states $S_1 \dots S_R$, and for each state generate the n-bit keystream sequence obtained by starting the KG in that state:

$$2^r \text{ subsequences} \begin{cases} S_1: & k_{1,1} & \cdots & k_{1,n} \\ S_2: & k_{2,1} & \cdots & k_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ S_R: & k_{R,1} & \dots & k_{R,n} \end{cases}$$

We then sort this list of subsequences (e.g. into lexicographic order); this requires storage of order $(n+r)2^r$ and time of order $(n+r^2)2^r$.

For a particular instance of an attack, we work through a sequence of known keystream, repeatedly extracting the next n-bit subsequence and checking our stored list for an exact match. Each attempt will require time of order r^2, and we expect to need roughly 2^{n-r} attempts before a match is found.

Choosing $r=n/2$, the attack will require a number of known keystream bits of order $2^{n/2}=\sqrt{|S|}$, time of order $n^2 2^{n/2} \approx \sqrt{|S|}$, and space of order $n2^{n/2} \approx \sqrt{|S|}$. Choosing $r>n/2$, i.e. doing more work in the pre-processing stage, an attack will require space of order $>n2^{n/2}$ and pre-processing time of order $>n^2 2^{n/2}$, but a number of known keystream bits of order $<2^{n/2}$ and actual attack time of order $<n^2 2^{n/2}$ — in principle the actual time for one attack can be made very small.

1.6 Non-Consecutive Keystream Bits

It is easy to see that the attack can be generalised to work with a sequence of regularly spaced rather than actually consecutive keystream bits (e.g. every 8th keystream bit), or indeed for regularly spaced linear combinations of keystream bits (e.g. the XOR of the 8 bits of each byte of keystream). This could make the attack particularly feasible in some cases.

2. Generalising to Other Stream Ciphers

The attack requires a sequence of known keystream bits. In many stream ciphers this is equivalent to a known plaintext attack — knowing a ciphertext symbol and the corresponding plaintext symbol allows the cryptanalyst to determine the keystream symbol. This is the case in a binary stream cipher where $c_t=p_t \oplus k_t$; and it is true more generally if $c_t=L(p_t,k_t)$, where L is any Latin square. But it is not generally true if $c_t = f_{k_t}(p_t)$, where $\{f_k\}$ is a set of invertible functions from the set of plaintext symbols to the set of ciphertext symbols.

3. Comparison with Earlier Results

Dodd [1] demonstrated an attack with very similar performance in the special case in which state transition is linear and the output function depends on $\leq n/2$ state bits. The attack presented in the present paper is much more generally applicable than Dodd's, but Dodd's attack can be more powerful in certain circumstances — it can use any known keystream bits (not just regularly spaced ones), it may be possible when some keystream bits are "known" only probabilistically (although more keystream bits and time will be needed), and if the output function depends on fewer than half of the state bits then far fewer keystream bits may be required.

4. Conclusion

The attack presented in this paper can improve the time taken to search through **all possible KG states**, not through all possible initial states (i.e. all possible secret keys). If the number of possible KG states is much greater than the number of possible secret keys, then the attack will provide no improvement.

From this we suggest a design principle for stream ciphers, to proof them against any attack of this sort: for a secret key length of i bits, a state size of $2i$ bits is desirable.

5. References

[1] Dodd, M.W., *Simultaneous Correlation to Many Linear Functionals: a New Cryptanalytic Technique which Can Almost Halve the Effective Key Size of Certain Stream Ciphers*, Proc. 4th IMA Conf. on Cryptography and Coding, Cirencester, 1993 (published by the IMA as "Codes and Cyphers: Cryptography and Coding IV", ed P.G.Farrell, 1995).