# Distributed Time-Memory Tradeoff Attacks on Ciphers

## (with Application to Stream Ciphers and Counter Mode)

Howard M. Heys

Department of Electrical and Computer Engineering
Memorial University of Newfoundland
St. John's, Newfoundland, Canada
email: hheys@mun.ca

**Abstract.** In this paper, we consider the implications of parallelizing time-memory tradeoff attacks using a large number of distributed processors. It is shown that Hellman's original tradeoff method and the Biryukov-Shamir attack on stream ciphers, which incorporates data into the tradeoff, can be effectively distributed to reduce both time and memory, while other approaches are less advantaged in a distributed approach. Distributed tradeoff attacks are specifically discussed as applied to stream ciphers and the counter mode operation of block ciphers, where their feasibility is considered in relation to distributed exhaustive key search. In particular, for counter mode with an unpredictable initial count, we show that distributed tradeoff attacks are applicable, but can be made infeasible if the entropy of the initial count is at least as large as the key. In general, the analyses of this paper illustrate the effectiveness of a distributed tradeoff approach and show that, when enough processors are involved in the attack, it is possible some systems, such as lightweight cipher implementations, may be practically susceptible to attack.

**Keywords:** cryptanalysis, time-memory tradeoff attacks, block ciphers, stream ciphers, counter mode

## 1 Introduction

Time-memory tradeoff (TMTO) attacks were first introduced by Hellman [1] to attack block ciphers using a chosen plaintext or easily predicted known plaintext. The basic concept involves two phases: Before system operation begins, the *preprocessing* (or offline) phase prepares a compact table from chains representing information from (almost) all keys, while the *online* phase efficiently searches the table in an attempt to identify which key is used to encrypt during system operation. Following Hellman's work, Babbage [2] and Golić [3] independently showed that a time-memory-data tradeoff based on the birthday paradox was applicable to stream ciphers by attacking the stream cipher state, rather than the key. This was subsequently combined with Hellman's approach by Biryukov and Shamir [4] to develop another, more flexible tradeoff involving data and targeting the stream cipher state. This approach was then extended by Hong and Sarkar [5] to attack directly the key and initialization vector (IV) of stream ciphers, as well as being applied to some block cipher modes.

Numerous papers have refined Hellman's approach trying various methods to improve the success rate and reduce the attack complexity. Most notably, the distinguished points method, attributed to Rivest in [6], can be used to minimize costly memory accesses, while

it is claimed that the rainbow table method can be used to minimize memory accesses and improve the speed of the table search [7].

Although the concept of distributed cryptanalytic attacks is well known, no paper has systematically characterized the value of distributed time-memory tradeoff attacks. In this paper, we examine tradeoff expressions for a number of distributed TMTO approaches using the number of processors as a tradeoff parameter. Further, we explicitly examine distributed attacks and their applicability to stream ciphers and the counter mode operation of block ciphers.

# 2 Background on Time-Memory Tradeoff Attacks

In this section, we provide detailed descriptions of TMTO attacks so that we have the foundation to later discuss the distributed TMTO attacks. The knowledgeable reader may wish to skip to Section 3.

In our discussion (as well as the discussions in other papers on TMTO attacks), complexities are given for time, memory, and data and the unit of these complexities may differ by a modest multiplicative constant when comparing approaches. Time and memory complexities are often represented in units equivalent to the number of encryption operations and the number of key pairs stored, respectively, while data complexities are sometimes expressed as the contiguous bits of data or the number of data blocks, with each block corresponding to a unique IV. Also, as is usually done, we assume that when an algorithm complexity involves a factor that is logarithmic in a parameter, this factor is small enough to be ignored.

A summary of the characteristics of tradeoff attacks, as well as the distributed versions discussed in this paper, is presented in Appendix A.

## 2.1 Hellman's Attack

The basic TMTO attack introduced by Hellman [1] works because memory is saved by storing in a table just the start and end of chains generated during the preprocessing phase, such that, in the online phase, the table can be efficiently searched while walking through a chain starting with the data captured from the system. As a result, the preprocessing phase requires a time complexity that is equivalent to the size of the key space, while the online time complexity and the memory complexity can be substantially less than the size of the key space.

Consider a block cipher with a block size of $b$ bits and a key space with a total of $K$ keys, where $K \leq 2^b$.[1] A plaintext block, $x$, is selected such that the corresponding ciphertext (resulting from encryption using the target key) can be captured during system operation. During the preprocessing phase, a chain is created by randomly selecting a key $k_0$ and encrypting plaintext $x$ using encryption function $E$ with key $k_0$ to produce ciphertext $y_1 = E_{k_0}(x)$. A chaining function $R$ is then applied to produce a value for the next key, $k_1$, from the ciphertext, $y_1$, so that $k_1 = R(y_1)$. In practice, $R$ can be a very simple function (such as a transposition on $\log_2 K$ bits selected from the $b$ bits of the block). Key $k_1$ is subsequently used to encrypt $x$ to produce $y_2 = E_{k_1}(x)$, which then has $R$ applied so that $k_2 = R(y_2)$. This process is repeated for a total of $t$ links in the chain, as illustrated in Figure 1, and the first key, $k_0$, and last key, $k_t$, are saved. A total of $m$ such chains for random starting keys are constructed with just the first and last keys being saved for each. For clarity, we shall refer to this table of $m$ key pairs as a *subtable*.

After the completion of the $m$ chains, the subtable of $m$ key pairs are sorted based on the last key in the chain. Hellman recommended the stopping criterion of $mt^2 = K$ as a constraint on the values of $m$ and $t$ to mitigate the merging of chains which causes

[1] The TMTO attack can also be easily applied when $K > 2^b$ using multiple blocks of plaintext for input.

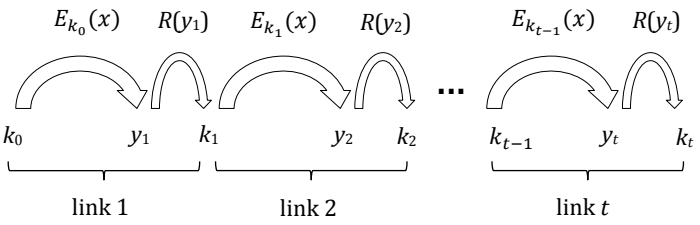


**Figure 1:** Chain in Hellman Table

repetition of keys covered by the subtable. Hence, one subtable only effectively covers about $K/t$ of the total of $K$ keys. To improve the coverage, Hellman suggested generating a table of $t$ such subtables, using different chaining functions $R_i$, $1 \leq i \leq t$, for each table. Since any two subtables have different chaining functions, even if a specific key value occurs in chains in both subtables, it will not result in the two chains merging. Since one subtable should cover about $mt$ key values, if $t$ subtables are constructed using different chaining functions, the resulting coverage of the table of $t$ subtables will be about $K$. This means that information on most of the keys is available from the table which has a memory complexity of $M = mt \ll K$, where a memory element contains a key pair corresponding to the first and last keys in a chain.

After the table is constructed in the preprocessing phase, in the online phase, the attacker acquires the ciphertext, $y$, corresponding to plaintext $x$ encrypted under the unknown target key value $k$, $y = E_k(x)$. For each subtable $i$, $1 \leq i \leq t$, after computing $R_i(y)$, the resulting value is looked up in the subtable, requiring a search of (insignificant) logarithmic complexity in $m$ since each of $t$ subtables is sorted by the last key of each chain. If this value is not found, the encryption is applied with the new key value given by $R_i(y)$, the chaining function applied, and the resulting value looked up in the subtable. This process is repeated (up to $t$ times) for each subtable. When a value is found in one of the subtables, the implication is that, with reasonable likelihood, the correct key is found in the corresponding chain and by restarting at the beginning of the chain (using the stored first key value) and continuing until $y$ is produced by the encryption in the chain, one can identify the target key as the key used to produce $y$ in the chain. The correctness of this key can be confirmed with other captured plaintext/ciphertext pairs. If a false alarm occurs, the search process continues and, with high probability, the correct key will eventually be found in the table. In the worst case, it will take $t$ encryptions of $x$ (to proceed through a chain) for each of $t$ subtables. Hence, the online time complexity is $T = t^2$, where the units of time are encryption operations.

Now, combining the stopping criterion of $mt^2 = K$, with the expressions for $M$ and $T$, it is simple to develop the Hellman tradeoff expression of

$$TM^2 = K^2. \tag{1}$$

The preprocessing time, $P$, required to set up the table must cover all key values and, hence, $P = K$. Hellman uses the example that, if $T = M$, then both online time and memory are smaller than the key space and, in fact, $T = M = K^{2/3}$.

## 2.2 Babbage-Golić (BG) Tradeoff

Both Babbage [2] and Golić [3] independently proposed a tradeoff attack on stream ciphers, referred to as the BG attack. Assume that the size of the stream cipher's state space is $N$. A keystream prefix is a $\log_2 N$ sequence of keystream bits corresponding to the state at which the prefix starts. The BG tradeoff works by constructing, during preprocessing,

a table of $N/D$ pairs of the state and the corresponding keystream prefix. A total of $D + \log_2 N - 1 \approx D$ bits of keystream are acquired in the online phase resulting in the determination of $D$ keystream prefixes, using a sliding window. Due to the birthday paradox, with high probablity, one of the $D$ keystream prefixes can be found in the table and the corresponding state derived, thus breaking the cipher.

For this attack, the tradeoff expression is

$$TM = N \tag{2}$$

where $T = D$, $M = N/D$, and the preprocessing time complexity is $P = N/D$. While a strong theoretical attack, the requirement for a large amount of data restricts the practicality of the attack.

Due to this attack, it is prudent to ensure that the state of the stream cipher (in bits) should be at least twice as large as the key (in bits) to ensure that $T \geq K$ or $M \geq K$. Thus, as long as $N \geq K^2$, the BG TMTO attack is not more efficient than exhaustive key search which has a time complexity of $K$.

## 2.3 Biryukov-Shamir (BS) Tradeoff

In [4], Biryukov and Shamir combined Hellman's table and the BG tradeoff use of data to develop a new tradeoff involving time, memory, and data, applicable to stream ciphers. In the BS tradeoff, the Hellman table is derived from chains on the cipher state, rather than the key. During preprocessing, a total of $t/D$ subtables are constructed using different chaining functions, with each covering $m$ chains of length $t$, for which only the first and last states are stored. Variable $D$ represents the amount of data in the form of contiguous keystream bits used in the attack and now $M = mt/D$. The preprocessing complexity is thus $P = N/D$, where $mt^2 = N$ is the stopping criterion for constructing the table.

During the online phase, $t$ steps through the chain (that is, state updates of the stream cipher, followed by the chaining function) must be executed, with each of the $t/D$ subtables being searched and this must be done for each of the $D$ prefixes derived from a sliding window over the $D$ bits. Hence, the online time takes $T = t(t/D)D = t^2$. As a result, the tradeoff in this case becomes

$$TM^2D^2 = N^2. \tag{3}$$

It should be noted that to ensure there is at least one complete subtable, it is assumed that $D \leq t$ and therefore the restriction of $D^2 \leq T$ exists. In order to be better than exhaustive search on the state, an attacker would select $T < N$ and $M < N$. However, since the state is determined by the key upon initialization, in order to be better than exhaustive key search, we expect $T < K$ and $M < K$, where $K$ represents the size of the key space. Hence, since $D^2 \leq T$, it can be seen that letting $N \geq K^2$ is sufficient to ensure that a BS TMTO attack cannot do better than exhaustive key search.

## 2.4 Hong-Sarkar (HS) Tradeoff

In [5], Hong and Sarkar explicitly relate the BS tradeoff for stream ciphers to the key and the IV, rather than the state. The key is secret and unknown when building the table during preprocessing and, while the IV is typically public and known during the online phase, it may be unpredictable and therefore also unknown when building the table during preprocessing. The HS tradeoff approach treats the input to be discovered in the tradeoff attack to be the key/IV combination. If the size of the IV space is defined to be $V$ and the IVs to be used by the system are unknown during preprocessing, then the HS approach can be applied to a stream cipher with the tradeoff being

$$TM^2D_{iv}^2 = (KV)^2 \tag{4}$$

where the preprocessing complexity is given by $P = KV/D_{iv}$. The attack has a similar data restriction of $D_{iv}^2 \leq T$ as the BS approach. Note that the $D$ term used in the BS tradeoff of (3) has been replaced by $D_{iv}$ in (4) to emphasize that, rather than $D$ contiguous bits, in fact, $D_{iv}$ represents the number of $\log_2(KV)$ bit prefixes at the start of the keystream for different key/IV combinations.

The HS tradeoff can be easily explained by noting that the state space defining the initial state following a reinitialization with a new IV cannot exceed the total space implied by the key/IV combination since the key/IV combination determines the initial state. It is possible for the HS tradeoff to be an improvement on the BS tradeoff in cases where $KV < N$, however, this is not necessarily the case. In practice, to apply the attack would require a lot more data to be collected than just $D_{iv}$ bits, although much of it could be discarded since it is not needed in online processing. This is discussed further in Section 4.3.

In theory, each prefix used in the attack must be collected from different key/IV combinations and, hence, success in the attack may mean finding one key from among a number of keys used in encryption. In the single-key scenario, where it is assumed that data is only available from one key, if unpredictable IVs are to be used, then data could be collected from different IVs and the target key. Then the tradeoff of (4) can be applied, where $D_{iv}$ represents the number of IVs under the one key and, hence, $D_{iv} \leq V$.

## 2.5 Dunkelman-Keller (DK) Approach

The HS tradeoff approach assumes that the preprocessing is structured to consider the combination of key and IV as one input and builds the table based on this, resulting in the restriction on data. However, the HS method of attack does not take advantage of the fact that, during the online phase, the IV is known and only the key needs to be discovered. In [8], Dunkelman and Keller modify the HS approach by separating the key and IV in the attack. The preprocessing phase then builds a number of Hellman tables to cover keys, with each table built for a particular IV. This allows the online phase of the attack to simply consider whether an intercepted IV has been used to build a table. If so, the table corresponding to this IV can be searched for the key. In this approach, which we refer to as the DK approach, assuming equally likely occurences of any IV, if $V/D_{iv}$ tables, each corresponding to a different IV, are built during preprocessing, then collected data from $D_{iv}$ IVs during the online phase, should result in one of the intercepted IVs being used in the tables with high probability. For this tradeoff, $M = (V/D_{iv})mt$ and $T = t^2$, where the stopping criterion of $mt^2 = K^2$ applies to the Hellman tables. Hence, the DK method has the tradeoff expression of (4) if the IV is unpredictable, but now has no restriction on the data, $D_{iv}$, other than $D_{iv} \leq V$ in the single-key scenario. Further, this approach has the advantage for applications where the IV is unpredictable but not equally likely in distribution, as this can be used to build tables for the IVs which are most likely to occur.

## 2.6 Practical Performance Issues for TMTO Attacks

We shall consider in our work both the distinguished points and rainbow table refinements of Hellman's TMTO attack. These refinements and their relative merits in terms of probability of success, detailed complexity analyses, and other practical performance related issues, are studied in a number of papers including [9][10][11]. The results of these comparisons indicate that these practical performance issues do not seem to have substantial implications (i.e., orders of magnitude effects on complexity) and, hence, we do not consider them significant for our discussion on distributed TMTO attacks.

### 2.7 Previous Work on Parallelized TMTO Attacks

It is known that it is possible to parallelize TMTO attacks. For example, distributed attacks are mentioned in [12] where it is noted that it is possible to divide the Hellman subtables into groupings and circulate to participating processors. Parallelizing TMTO attacks is further studied in [13][14]. However, no work has yet systematically characterized the tradeoff aspects of multiple processors. In our work, we will thoroughly characterize the distributed approach to various forms of time-memory tradeoffs. As a point of comparison for distributed tradeoff attacks, we consider distributed exhaustive key search and dictionary attacks, which represent the extreme ends of the tradeoff spectrum of "large time complexity / negligible memory complexity" and "negligible time complexity / large memory compexity", respectively. The reader is referred to Appendix B for a discussion of the distributed versions of these basic cryptanalytic attacks.

## 3 Distributed Hellman Attack

We now consider the parallelization of Hellman's attack using distributed processors, as well as the related approaches of distinguished points and the rainbow table. We assume that $W$ processors, with independent memory, are available. This might represent, for example, $W$ computers on the Internet with users willing to participate, or being duped into participating, in attacking some cryptographic system. We assume that any necessary communication complexity between these processors and a central controlling processor are negligible in comparison to the time and memory complexities associated with the attack.

In our discussion, we let $T_0$, $M_0$, and $P_0$ represent the online time complexity, memory complexity, and preprocessing time complexity, respectively, for an individual processor. It is these quantities, along with $W$, which determine the efficacy of the attack, since it is assumed that the individual processors can operate concurrently. For example, while a non-distributed attack might require an online time complexity of $T$, if it is possible to spread this work evenly between $W$ processors, each processor would only require a time of $T/W$, which could be done concurrently for all processors, and thus the overall duration of the attack could be dramatically reduced if $W$ is large.

### 3.1 Distributed Approach to the Original Hellman Attack

A distributed approach to Hellman's TMTO attack can proceed by distributing the responsibility for generating the $t$ subtables to the $W$ processors, so that each processor generates $t/W$ subtables independently. When the necessary ciphertext data is captured during system operation, it will be distributed to all processors. Each processor will require a memory of $M_0$, where $M_0 = m(t/W) = M/W$ and $M$ is the total memory requirement for the attack, with $W \leq t$ in order to ensure that each processor generates one or more subtables.

Since each processor only needs to implement $t$ encryptions for each of $t/W$ subtables, the time taken in a processor (and, if all processors operate concurrently, the overall time to search the full Hellman table) is $T_0 = t(t/W) = T/W$, where $T$ is the time required for the non-distributed attack. When a key is found by a processor in its share of the table, it must communicate this back to the central processor that is overseeing the cryptanalytic process and that will be able to announce the successful completion of the attack.

Now $T_0 M_0^2 = (t^2/W)(mt/W)^2 = (mt^2)^2/W^3$ and assuming the Hellman stopping criterion of $mt^2 = K$ results in the tradeoff for an individual processor to be

$$T_0 M_0^2 W^3 = K^2 \tag{5}$$

where the constraint $W \leq t$, or equivalently $W \leq T_0$, applies. This expression captures the tradeoff of interest in a distributed Hellman attack and reflects that both time and memory can be improved by a factor of $W$. The preprocessing time for an individual processor is $P_0 = K/W$ and is improved by a factor of $W$ over the time required in the non-distributed attack, since each processor only needs to construct chains covering a fraction of the table. Although we notate this as the preprocessing cost of the individual processor, if we assume that all processors compute their tables concurrently, it also reflects the overall time complexity to prepare for the attack.

It is clear that using a number of processors to implement the attack potentially provides a very significant advantage and may actually make the attack possible in some practical scenarios. Of course, similar statements can be made about exhaustive key search or a dictionary attack: A distributed exhaustive key search can be sped up by a factor of $W$, while a distributed dictionary attack can reduce the memory requirement of an individual processor by a factor of $W$. (See Appendix B for a discussion of these attacks.) However, a distributed TMTO attack preserves the possibility for a significantly faster online processing time at the expense of more memory, or vice versa.

Consider the following example applying to an implementation of AES-128 for which $K = 2^{128}$. Letting $W = 2^{20}$, the non-distributed exhaustive key search would require $T = 2^{128}$, while the distributed exhaustive key search would require $T_0 = 2^{108}$. In the case of a Hellman TMTO attack with equal online time and memory complexity, the non-distributed attack would take $T = M = 2^{85.3}$ (with $P = 2^{128}$), while the distributed approach would require $T_0 = M_0 = 2^{65.3}$ (with $P_0 = 2^{108}$). As another example, consider a lightweight block cipher with an 80-bit key so that $K = 2^{80}$. In this case, with $W = 2^{20}$, a distributed TMTO attack exists with $T_0 = M_0 = 2^{33.3}$ (and $P_0 = 2^{60}$), which is substantially less complex than the $T_0 = 2^{60}$ required for a distributed exhaustive key search.

## 3.2 Distributed Distinguished Points (DP) Method

One of the issues identified for the Hellman TMTO attack is that the cost of a memory access can be highly variable depending on whether the access is to internal memory (RAM) or to an external memory (eg. hard disk drive or a solid state drive). For example, the cost of accessing an external drive may be orders of magnitude more than an internal RAM access [15]. In developing the tradeoff expression of (1), it was assumed that the time to access memory is negligible in comparison to an encryption operation and, hence, time complexity is enumerated relative to the number of encryption operations. However, if this assumption does not hold true, then in practical terms the attack may run much slower than anticipated. In order to mitigate the cost of slow memory accesses, the distinguished points (DP) method was proposed by Rivest [6]. In this approach, rather than build chains of fixed length $t$ when constructing a Hellman table, the preprocessing phase can build a chain which terminates when a particular pattern (eg. all zeroes) is recognized in the first $\log_2 t$ bits of the key. This means the length of a chain is variable but will be a length of $t$ on average. When executing the online portion of the attack, since the end point of a chain must start with $\log_2 t$ zeroes, only about $1/t$ encryptions needs a look up to be executed in the subtable (which is likely stored in slow access external memory). Hence, the number of memory accesses is reduced by a factor of $t$.

In the distributed Hellman attack, it is fully possible to execute the distinguished points approach to the attack. The amount of memory in a processor is still fixed at $M_0 = mt/W$, since there are $t$ subtables split between the $W$ processors. However, the time required to finish the concurrent computations of $W$ processors is now more complex. Since there is an average of $t$ links in each chain, the number of encryptions per subtable must be more than $t$ to cope with chains having longer than $t$ links. Assume that, at most, $\gamma t$ encryptions are executed for each subtable. The DP method is likely to set $\gamma$ to be a modest value,

to keep the time complexity of the attack constrained. When preparing the table during the preprocessing phase, the DP method will stop a chain when a distinguished point is found or when $\gamma t$ links in a chain have been reached without hitting a distinguished point. Similarly, during the online process, if, after $\gamma t$ encryptions, a distinguished point is not reached for a subtable, the subtable is assumed to not contain the key. Of course, the value used for $\gamma$ affects the probability of success, but as shown in [10], $\gamma$ can effectively be a small constant. Hence, the online time complexity can be no worse than the maximum chain length, $\gamma t$, multiplied by the number of subtables to search through, $t/W$, and, hence, $T_0 = \gamma t^2/W$ where $T_0$ now represents the maximum possible time taken at an individual processor.

This leads to a tradeoff of the form $T_0 M_0^2 W^3 = \gamma K^2$ which is slightly worse than the distributed Hellman tradeoff of (5). However, it is quite possible that implementing the distinguished points method when using a distributed approach will not be necessary. Since the memory size needed in the individual processors in a distributed attack is reduced by a factor of $W$, it is quite conceivable for some parameters that the processor memory complexity of $M_0$ is small enough that the processor's complete table portion could be stored in internal memory and slow accesses to external memory are not needed. In such a case, there would be no need to implement the DP approach.

## 3.3 Distributed Rainbow Table Method

In [7], Oechslin proposed an alternate formulation to represent the key chains in the TMTO attack. Hellman's approach was to use one chaining function for every link in a chain and for all the chains in one subtable, with different subtables then using different chaining functions. In contrast, the rainbow table approach uses a different chaining function for each link in the chain and then builds one table of such chains. That is, link $i$, $1 \leq i \leq t$, in each chain uses function $R_i$. It is shown that the same relationship of $mt^2 = K$ should be used and the result is one table covering $M = mt = K/t$ chains of length $t$. It is argued that there are improvements to Hellman's approach [7][16]. For the online phase, the number of encryptions is determined by computing $t$ partial chains, where the $j$-th partial chain is of the form

$$R_{t+1-j}(\cdot) \rightarrow E_{(\cdot)}(x) \rightarrow R_{t+2-j}(\cdot) \rightarrow E_{(\cdot)}(x) \rightarrow ... \rightarrow R_{t-1}(\cdot) \rightarrow E_{(\cdot)}(x) \rightarrow R_t(\cdot) \qquad (6)$$

resulting in the total number of encryptions to be about $t^2/2$. Ignoring the somewhat insignificant factor of $1/2$ in the number of encryptions gives $T \approx t^2$ and results in the same tradeoff expression as in (1). However, since only at the end of one of the partial chains is it necessary to look up in the table, only $t$ memory accesses to the table are required.

The distributed rainbow table approach can be accomplished by distributing the table so that $M_0 = mt/W = M/W$. However, for each processor, the time complexity involves reproducing $t$ partial chains for a total of $T_0 = t^2/2 \approx t^2$ encryptions required in each processor. Hence, the time complexity cannot be improved by distributing the table since each processor must take $\sim t^2$ to consider their portion of the table, i.e., $T_0 = T$. The resulting tradeoff expression is

$$T_0 M_0^2 W^2 = K^2. \qquad (7)$$

Rather than divide up the rainbow table between processors, an alternative approach for a distributed rainbow table attack, would be to distribute the computation of $t$ partial chains between $W$ processors. In this case, $T \approx t(t/W)$ would represent the online time complexity, where we use the approximation sign to indicate that partial chains have about $t/2$ encryptions on average. However, the resulting distributed computations would need to be checked in one central table. In this case, $T_0 = T/W$, but $M_0 = M = mt$. Hence,

the tradeoff becomes even worse as

$$T_0 M_0^2 W = K^2. \tag{8}$$

For the rainbow table approach, distributing the table and the computations is not feasible, since the end of each partial chain must be looked up in the full table. Comparing (7) and (8) to the tradeoff of (5) representing the distributed Hellman attack, it is obvious that the tradeoff of the distributed rainbow table approach is inferior to the distributed version of the original Hellman TMTO approach. In addition, when applying a distributed approach to time-memory tradeoffs, since the memory requirements could be substantially smaller on a per processor basis, reducing memory accesses (one of the advantages of the rainbow table) may very well not be important, since the necessary subtables of the Hellman approach may fit within a processor's RAM.

# 4 Applying Distributed TMTO Attacks on Stream Ciphers

In this section, we consider the application of distributed TMTO attacks to stream ciphers.

## 4.1 Distributed BG Attack

We first consider the distributed BG attack, which makes use of data collected and assumes $D$ bits of keystream are available. In this case, the attack can be distributed by dividing up the work to prepare, and the memory to store, the BG table to $W$ processors, so that $P_0 = N/(DW)$ and $M_0 = N/(DW)$. The time required in a processor during the online phase is directly proportional to the processing of all $D$ prefixes, so that $T_0 = D$, which is unchanged from the non-distributed case. As a result, it can be shown that

$$T_0 M_0 W = N. \tag{9}$$

Consider the general case where $T_0 = M_0^r$, $r > 0$. Then, it can be shown that $T_0 = (N/W)^{r/(r+1)}$. Since the exponent is always less than 1, $T_0 < N/W$, which is equivalent to saying that the distributed BG tradeoff always leads to an attack which has a better online time complexity than distributed exhaustive search on the state of the stream cipher. (Similar reasoning applies to $M_0$.)

For a non-distributed attack, letting $N \geq K^2$ ensures that the BG tradeoff does not lead to a better attack than exhaustive key search. Placing this constraint on the stream cipher leads to the following proposition for the distributed BG attack.

**Proposition 1**
If $N \geq K^2$, there is no value of $W$ for which a distributed BG TMTO attack on a stream cipher has a lower complexity for both online time and memory than the complexity of distributed exhaustive key search.

**Proof**
A distributed exhaustive key search has a complexity of $K/W$. Let $N = aK^2$, where $a \geq 1$. We can now adjust (9) to be $T_0 M_0 W = aK^2$. For the best TMTO attack, we can minimize the maximum of either $T_0$ or $M_0$ in this equation by letting $T_0 = M_0$, leading to

$$T_0 = \frac{a^{1/2} K}{W^{1/2}} \tag{10}$$

which clearly implies $T_0 \geq K/W$ and $M_0 \geq K/W$ for all values of $W$. Since other tradeoffs lead to one of $T_0$ or $M_0$ being larger, there will always be at least one of $T_0$ or $M_0$ being at least as large as $K/W$. Hence, clearly the distributed BG tradeoff cannot have a lower

complexity than distributed exhaustive key search for any number of processors. □

As an example, consider Grain v1 [17], for which the size of the stream cipher state is 160 bits and the key is 80 bits. Using (9) and letting $T_0 = M_0$, if $W = 2^{20}$, then $T_0 = M_0 = 2^{70}$, which is better than a non-distributed exhaustive key search and non-distributed BG TMTO for which, in both cases, $T = 2^{80}$. However, for distributed exhaustive key search, $T_0 = 2^{60}$. In general, when $N = K^2$, in comparison to a non-distributed attack, the distributed BG attack only improves time complexity by a factor of $W^{1/2}$ over non-distributed exhaustive key search, whereas the distributed exhaustive key search and the distributed Hellman attack improve by a factor of $W$.

## 4.2 Distributed BS Attack

Consider now the distributed BS attack. With $W$ processors and $D$ contiguous data bits of keystream, the $t/D$ subtables needed in the BS approach can be divided into $W$ groups, resulting in the memory for individual processors being $M_0 = mt/(DW)$, where $W \leq t/D$ in order for each processor to have one or more subtables. The time in an individual processor to process the data and recover the state is given by $T_0 = t{\cdot}(t/(DW)){\cdot}D = t^2/W$, where the first term represents the $t$ encryptions to reproduce a chain from the starting point of the captured data, the middle bracketed term represents the number of subtables to process in each processor, and the last term represents the data that each processor must consider. Combining the expressions for $M_0$ and $T_0$ leads to the following tradeoff:

$$T_0 M_0^2 D^2 W^3 = N^2 \tag{11}$$

where the amount of data and the number of processors must satisfy $D^2W \leq T_0$ (which is derived by combining the constraint on $W$ with the expression for $T_0$). Since deriving the required subtables determines the preprocessing time in an individual processor, we also have $P_0 = N/(DW)$.

For a distributed BS attack, the constraint of $N \geq K^2$ ensures that the distributed BS tradeoff performs no better than distributed exhaustive key search. We illustrate this formally in the following proposition.

**Proposition 2**
If $N \geq K^2$, there is no value of $W$ for which a distributed BS TMTO attack on a stream cipher, satisfying the constraint $D^2W \leq T_0$, has a lower complexity for both online time and memory than the complexity of distributed exhaustive key search.

**Proof**
Let $N = aK^2$, where $a \geq 1$. Minimizing $T_0$ and $M_0$ in the application of the BS tradeoff is done by maximizing the data in the tradeoff. Using the upper bound of $D \leq (T_0/W)^{1/2}$, it can be shown that (11) is equivalent to the tradeoff of $T_0 M_0 W = aK^2$. This is now identical in form to the distributed BG tradeoff of (9) and, hence, the remainder of the proof can follow similarly to the proof of Proposition 1. □

Using the tradeoff of (11) could be useful in circumstances where the ratio of the state size in bits to the key in bits is less than 2. Consider that, if $N = 2^{160}$ and $D = 2^{25}$, then letting $T_0 = M_0$, one could use $W = 2^{20}$ processors to achieve $T_0 = M_0 = 2^{70}$, where $D^2W \leq T_0$ is satisfied. In comparison, this is clearly better than the distributed BG attack which would require $T_0 = D = 2^{25}$ and $M_0 = 2^{115}$. Further, a distributed BS TMTO attack would be more successful than a distributed exhaustive key search, if the key size was such that the online time complexity for the BS tradeoff satisfies $T_0 < K/W$ and this is true here if the key size is more than 90 bits.

### 4.3 Distributed HS and DK Attacks

Targeting a stream cipher system which uses a single key and numerous IVs and applying a distributed HS approach results in the tradeoff

$$T_0 M_0^2 D_{iv}^2 W^3 = (KV)^2, \tag{12}$$

where $D_{iv}$ represents the number of prefixes that are derived from the first $\log_2(KV)$ bits of the initial cipher state following the reinitialization from different IVs. The constraints $D_{iv}^2 W \leq T_0$ and $D_{iv} \leq V$ apply and the preprocessing complexity is $P_0 = (KV)/(D_{iv}W)$. (It is also conceivable to attack the cipher obtaining data from different keys, however, we focus on the single key scenario, thereby assuming that all data collected and used in the attack comes from one key.)

The distributed DK approach, which builds $V/D_{iv}$ Hellman tables for different IVs results in the same tradeoff as (12), as well as the same constraint of $D_{iv} \leq V$ and the same preprocessing complexity of $P_0 = (KV)/(D_{iv}W)$. However, since the DK approach builds a Hellman table to cover just keys (rather than key/IV combinations), we can assume that each processor contains $t/W$ of the Hellman subtables for all of the $V/D_{iv}$ IVs. In this case, $M_0 = (V/D_{iv})m(t/W)$ and $T_0 = t(t/W)$, resulting in (12) with the contraint that $W \leq t$, or equivalently $W \leq T_0$, since at least one full subtable per IV must be stored in a processor.

It appears that, if $KV < N$, the distributed HS and DK formulations are preferred over the BS approach. However, comparing (11) and (12) can be deceiving since the data term in both expressions represents different types of data. In the case of the BS approach of (11), the data represents the total number of contiguous bits collected under the same key and IV. Conversely, the HS and DK approaches of (12) require a total number of bits of data to be about $D_{total} = D_{iv}\mu_{iv}$, where $\mu_{iv}$ represents the average number of bits encrypted under one IV (although only the first $\log_2(KV)$ bits of each IV's keystream are used in the attack). Hence, substituting into (12) results in

$$TM^2 D_{total}^2 W^3 = (KV\mu_{iv})^2 \tag{13}$$

where $D_{total}$ is the number of bits collected (although many are discarded) and, while it represents data collected from multiple IVs, it is similar to the $D$ term in (11), implying that (12) is a better tradeoff (with a smaller term on the right), when $KV\mu_{iv} < N$. In cases where $N = K^2$, which ensures security against BG and BS attacks and minimizes cipher implementation complexity, (12) is the better tradeoff when $V\mu_{iv} < K$. These same arguments apply equally to the non-distributed and distributed HS and DK approaches.

As an example, consider Grain v1 [17] with state size of 160 bits, key size of 80 bits, and IV size of 64 bits. As long as the average number of plaintext bits encrypted for each IV is less than $2^{16}$, the HS and DK attack formulations are better than attacking the state of the cipher using the BS tradeoff. Note that if the IVs used in a stream cipher system are not equally likely, then the DK approach can effectively build tables for the most likely IVs and can be more successful than the HS approach in both the non-distributed and distributed cases.

## 5 Applying TMTO Attacks to Counter Mode

In this section, we describe how non-distributed and distributed TMTO attacks can be applied to counter mode [18]. We focus our attention on counter mode because TMTO attacks have not been fully explored for their applicability to the mode. Other modes (such as ECB) can be straightforwardly attacked using chosen or (predictable) known plaintext approaches and applying TMTO attacks such as Hellman's original tradeoff or its distributed version. However, when a block cipher operates in counter mode, in addition

to the key, the count value can be initialized so that the count is unpredictable during the preprocessing phase of TMTO attacks, making the building of the Hellman table more challenging. This implies that TMTO attacks can be made more difficult, even when a chosen plaintext approach can be applied during the online phase. In this section, we explore the viability of increasing the security of counter mode against TMTO attacks by making the initial count unpredictable.

## 5.1 Counter Mode Model

When used in counter mode, a keyed block cipher encrypts a count value held in a $b$-bit counter register, where the count is incremented by a simple update function such as adding 1 to a binary number or applying LFSR functionality. During the operation of the system under a given key, the count is periodically initialized to a particular value that we shall refer to as the *initial count*. It is important not to repeat the count under a given key because this will result in a repetition of the keystream. We shall consider a format for the initial count that divides the $b$ bits into two parts: (1) a $v$-bit initialization vector (IV) or nonce[2] and (2) a $c$-bit block counter field, where $b = v + c$. The IV field is used to ensure that no counts are repeated for a given key and the IV could be (i) a predictable, unique value (eg. message number) or (ii) a randomly selected, unpredictable value. Although the IV (and hence the initial count) may not be predictable, it is not usually kept secret and is typically publicly exchanged between parties at the start of a communication session or message.[3] The $c$-bit block counter field increments for every block of encryption. We assume that it will be typically initialized to a known value (such as all zeroes) for every new IV.

When counter mode is operated with a non-random, predictable IV, along with a known initial value for the block counter field, Hellman's TMTO attack can be directly applied by constructing tables for the predicted initial count. The distributed attack can be applied as described in Section 3.

In the rest of the paper, we consider counter mode with a random, unpredictable IV component in the initial count. Although it is still assumed that the initial count value can be intercepted and known to the attacker during the online phase of a TMTO attack, we assume that all $V = 2^v$ values of IV are random and equally likely when building the tables during the preprocessing phase. This is equivalent to having the entropy of the initial count being $\log_2 V = v$.

## 5.2 Non-Distributed TMTO Attack on Counter Mode

In this section, we consider first the application of Hellman's original, non-distributed TMTO attack to counter mode. In all cases in this section, we focus the discussion on extracting the key from a system which uses a single key. The multi-key scenario is discussed in Section 5.5.

When the IV is unpredictable, the initial count value is not known before the use of the cipher and it cannot be used to build the Hellman table to cover the keys. We assume therefore that it is not known how to build a table for any resulting value in the counter register. One approach, if $V$ is small enough, could therefore be to build Hellman tables for keys, corresponding to all $V = 2^v$ values. Once an IV is used and observed, the appropriate Hellman table can be accessed and the TMTO attack executed. In this case, the memory

[2] We use the terminology "IV" to refer to the field of the initial count that must be unique for a given key in order to be consistent with terminology used in the stream cipher domain. In some contexts, this field may be referred to as a nonce.

[3] We will sometimes refer to the IV, when more precisely, we should refer to the initial count that is intercepted and collected. Strictly, we consider the IV to be the random, unpredictable part of the initial count but, for convenience, we often treat the IV as synonymous with the initial count.

required is $M = Vmt$ and the time required for the attack is $T = t^2$, resulting in tradeoff

$$TM^2 = (KV)^2 \tag{14}$$

based on the Hellman stopping criterion of $mt^2 = K$. The preprocessing complexity is $P = KV$. Clearly, the tradeoff attack is worse than Hellman's attack on the cipher in ECB mode and the preprocessing stage is more costly than exhaustive key search since $P > K$. Also, if $V > t$, the memory requirement is worse than the full dictionary attack, since this implies $M > K$. However, for some relative values of $K$ and $V$, the online phase of the TMTO attack can be better than exhaustive key search.

Note that this is the same tradeoff expression that would apply if we used the TMTO attack on a function with an input that is the combination of the key and IV, resulting in an input space of $KV$. In counter mode, determining the IV in the attack has no value since it is assumed to be intercepted and known during the online phase.

It is not hard to show from (14) that if $V \geq K^{1/2}$, at least one of $T$ or $M$ must be at least as large as the complexity of exhaustive key search. Hence, in order to ensure that a non-distributed TMTO attack (which does not use multiple data) can be no better than exhaustive key search, the size of the IV field of the initial count, should be at least half the size of the key, that is, $v \geq k/2$, where $K = 2^k$. This has implications for practical specifications of counter mode. For example, using AES-128 (where $K = 2^{128}$) in counter mode with an unpredictable IV so that $V = 2^{32}$, since $V < K^{1/2}$, it is possible to mount a TMTO attack with an online time or memory complexity better than $2^{128}$. In fact, a TMTO attack can be mounted with $T = M = (KV)^{2/3} = 2^{106.67}$. The preprocessing phase of the attack has a complexity of $P = KV = 2^{160}$.

The TMTO attack can be improved by making use of data using the DK approach. Allowing for the collection of data from $D_{iv}$ IVs, we could implement only $V/D_{iv}$ of the total $V$ Hellman tables. Each captured IV can be checked to see whether it is used for one of the tables. This can be assumed to take negligible time if the IV is not used. However, with high probability, we would expect one of the collected IVs to be used for a table and the attack on the key can proceed with the appropriate table. In this case, $M = (V/D_{iv})mt$ since $V/D_{iv}$ Hellman tables are used. Since the online time complexity is dominated by the time taken for an IV used in the tables, $T = t^2$. The preprocessing complexity is now $P = KV/D_{iv}$ and the tradeoff expression is given by (4), which is derived using the stopping criterion of $mt^2 = K^2$.

Of course, $D_{iv} \leq V$ and, in practice, the number of data bits collected is required to be more than $D_{iv}$, since many plaintext blocks will be encrypted for a given IV. This extra data can be ignored in the attack as it does not affect either the amount of memory needed or the time complexity. For $K = 2^{128}$ and $V = 2^{32}$, if $2^{20}$ initial count values can be collected corresponding to $2^{20}$ different IV values, it is possible to mount a TMTO attack with $T = M = 2^{93.3}$ using $P = 2^{140}$.

It is clear that, since $D_{iv} \leq V$, $P \geq K$ and $TM^2 \geq K^2$. Hence, even using data, the preprocessing step is at least as slow as exhaustive key search and the tradeoff is worse than the Hellman tradeoff attack on ECB mode as in (1). However, we note that the Hellman tradeoff of (1) cannot be directly applied to counter mode unless the counter register content can be predicted during the preprocessing phase. To improve on the attack, we can apply a distributed approach and this will be considered in the next section.

## 5.3 Distributed TMTO Attack on Counter Mode

In this section, we consider the application of a distributed TMTO attack to counter mode with a single key and an unpredictable IV. For an attack which does not use data in the tradeoff, this can be done by dividing the $t$ subtables of $V$ Hellman tables (one table for each IV, covering all keys) between the $W$ processors. The tradeoff used in this approach

would be a simple modification of (5), where $K$ is replaced by $KV$:

$$T_0 M_0^2 W^3 = (KV)^2 \tag{15}$$

with $W \leq T_0$ and preprocessing requiring $P_0 = KV/W$ to cover all key/IV combinations across all processors. We now consider the development of an expression which indicates the size of $W$ necessary to allow a TMTO attack to outperform a distributed exhaustive key search. This is equivalent to saying that the online time complexity and memory complexity of the TMTO attack should be both less than $K/W$. The resulting analysis leads to Proposition 3.

**Proposition 3**
Consider counter mode such that the key and the IV portion of the initial count are unpredictable during the preprocessing phase and assumed to be randomly drawn from the $K$ and $V$ possible values, respectively. With $T_0 = M_0^r$, a distributed tradeoff approach can be applied to obtain an attack with an online time complexity and memory complexity less than the complexity of distributed exhaustive key search for the following conditions on $W$:

$$W > \begin{cases} V^{\frac{2}{1-r}}/K^{\frac{r}{1-r}} & , r < 1 \\ 0 & , r = 1, \text{ if } V < K^{1/2} \\ \infty & , r = 1, \text{ if } V \geq K^{1/2} \\ K^{\frac{r-2}{2r-2}} V^{\frac{2r}{2r-2}} & , r > 1 \end{cases} \tag{16}$$

**Proof**
We need to show the conditions on $W$ for which $T_0 < K/W$ and $M_0 < K/W$. The proof considers the three cases for $r$. For $r > 1$, $T_0 > M_0$ and, hence, it is sufficient to consider scenarios for $T_0 < K/W$, while for $r < 1$, $M_0 > T_0$, and, therefore, it is sufficient to consider $M_0 < K/W$. For the case of $r = 1$, $T_0 = M_0$ and we can consider a bound on either $T_0$ or $M_0$.

From (15), it can be shown that, if $r > 1$, then

$$T_0 = \frac{(KV)^{\frac{2r}{r+2}}}{W^{\frac{3r}{r+2}}} \tag{17}$$

which, when letting $T_0 < K/W$, leads to the case for $r > 1$.

Similarly, for $r < 1$,

$$M_0 = \frac{(KV)^{\frac{2}{r+2}}}{W^{\frac{3}{r+2}}} \tag{18}$$

which, when letting $M_0 < K/W$, leads to the case for $r < 1$.

Finally, letting $T_0 = M_0$, gives

$$T_0 = \frac{(KV)^{2/3}}{W} \tag{19}$$

which, when compared to $K/W$, results in an inequality not involving $W$, but which shows that, for $V < K^{1/2}$, the TMTO attack can improve upon exhaustive search for any $W$, while, for $V \geq K^{1/2}$, the TMTO attack cannot improve upon exhaustive search for any $W$.□

The interpretation of Proposition 3 can be demonstrated by considering the following example where we let $K = 2^{128}$ and $V = 2^{32}$. From Proposition 3, we can determine: (1) if $T = M$, then $W > 0$, (2) if $T = M^{1/2}$, then $W > 1$, and (3) if $T = M^2$, $W > 2^{64}$. So we can conclude that a distributed TMTO attack can be made more efficient than distributed exhaustive key search for cases 1 and 2 by using as few as 1 and 2 processors, respectively,

while for case 3, the number of processors must be more than $2^{64}$, an impractically large requirement. Hence, for case 3, although it may be theoretically possible to mount a distributed TMTO attack, it is not practical to do so. Other examples for values of $K$, $V$ and $r$ can be considered to determine their practicality in terms of the number of required processors in a distributed attack.

We have seen for the case of $r = 1$ that if the condition $V \geq K^{1/2}$ is satisfied, then any value of $W$ cannot lead to the online time and memory complexity of the TMTO attack as an improvement over exhaustive key search. We can now develop Proposition 4 which gives the relationship between $K$ and $V$ in order to ensure that it is impossible for a distributed TMTO attack to outperform distributed exhaustive key search for any tradeoff of time and memory.

**Proposition 4**
Consider counter mode such that the key and the IV portion of the initial count are unpredictable during the preprocessing phase and assumed to be randomly drawn from the $K$ and $V$ possible values, respectively. If $V \geq K^{1/2}$, the online time complexity or the memory complexity of a distributed TMTO attack (which does not use multiple data) is at least as large as the complexity of a distributed exhaustive key search.

**Proof**
The best tradeoff from (15) occurs when we minimize the maximum of either $T_0$ or $M_0$, which occurs for $T_0 = M_0$, leading to $T_0 = (KV)^{2/3}/W$. If $V \geq K^{1/2}$, in this case clearly $T_0 \geq K/W$ and $M_0 \geq K/W$ for any $W$, where $K/W$ is the complexity of a distributed exhaustive key search. Reducing $T_0$ at the expense of $M_0$ (or vice versa) would still clearly result in $M_0$ (or $T_0$) being at least $K/W$. □

Proposition 4 states that a distributed TMTO attack (which does not use multiple data) cannot be made more efficient than a distributed exhaustive key search for $V \geq K^{1/2}$ and this was argued to be the case for the non-distributed attack as well. The broad implication is that to ensure security against TMTO attacks (which do not use multiple data), the unpredictable portion of the initial count (which is equivalent to the entropy of the initial count) should be at least half the size of the key.

## 5.4 Incorporating Data into the Distributed Attack on Counter Mode

Consider now incorporating the use of data into the distributed TMTO attack on a single-key implementation of counter mode. In doing so, the distributed DK approach can be applied and, hence, the tradeoff of (12) can be used, with the constraints $W \leq T_0$ and $D_{iv} \leq V$, and $P_0 = KV/(D_{iv}W)$.

We can extend Proposition 4 to apply to TMTO attacks which use data, resulting in the following proposition.

**Proposition 5**
Consider counter mode such that the key and the IV portion of the initial count are unpredictable during the preprocessing phase and assumed to be randomly drawn from the $K$ and $V$ possible values, respectively. Assume that a distributed TMTO attack on a single-key system is applied with data available from $D_{iv}$ IVs, where $D_{iv} \leq V$. If $V/D_{iv} \geq K^{1/2}$, the online time complexity or the memory complexity of a distributed TMTO attack is at least as large as the complexity of a distributed exhaustive key search.

**Proof**
We can simply follow the proof of Proposition 4, but base it on the distributed DK tradeoff

of (12), which can be rewritten to be

$$T_0 M_0^2 W^3 = (K[V/D_{iv}])^2. \tag{20}$$

This equation is similar to (15) used in the proof of Proposition 4, except that we have substituted $V$ with $V/D_{iv}$. Proposition 4 now follows with the same substitution, resulting in the distributed TMTO attack with data not being able to improve on distributed exhaustive key search when $V/D_{iv} \geq K^{1/2}$. □

Proposition 5 increases the lower bound on $V$ for which the distributed TMTO attack becomes infeasible. The same bound can be trivially shown to be applicable to the non-distributed TMTO attack using data. Assuming that it is impractical for $D_{iv} > K^{1/2}$, then letting $V \geq K$ is sufficient to ensure security against TMTO attacks which make use of data. (Note that the requirement that $V \geq K$ was also suggested in [5] in the context of non-distributed attacks on stream ciphers and allowing $D_{iv} = K^{1/2}$. In [8], the requirement of $V \geq K^{3/2}$ is recommended based on non-distributed TMTO attacks on stream ciphers and allowing $D_{iv} = K$, which may be impractical. Hence, although $V \geq K$ is sufficient based a reasonable expectation on the amount of data that could be used, a cautious cryptographer may wish to ensure that $V \geq K^{3/2}$.)

Now if $D_{iv}W = \alpha V$, where $\alpha > 1$, then $P_0 < K$, meaning the preprocessing time is better than exhaustive search on a cipher with key space $K$. Further, $T_0 M_0^2 = K^2/(\alpha^2 W) < K^2/W$, which could be substantially better than the tradeoff of the non-distributed approach. Consider the following case of counter mode using AES-128: $K = 2^{128}$, $V = 2^{32}$ and $W = 2^{20}$. If we let $T_0 = M_0$ and $D_{iv} = 2^{20}$ (so that $\alpha = 256$), we get $T_0 = M_0 = 2^{73.3}$, with $P_0 = 2^{120}$. Hence, the complexity of the online phase of the distributed TMTO attack is much better than the complexity of distributed exhaustive key search, which would be $K/W = 2^{108}$. Of course, collecting more data $D_{iv}$ and/or involving more processors $W$ could be used to improve the attack even further, but is still subject to the DK approach constraints of $D_{iv} \leq V$ and $W \leq T_0$.

### 5.5 Applying Multiple Keys in the Attack on Counter Mode

In [5][19], the concept of attacking a multi-key block cipher system is discussed, where success in the attack is considered to occur if one of many used keys is found. Using data from different keys, it is possible to make use of the BS tradeoff expression of (3) in attacking block ciphers in ECB mode by collecting $D$ ciphertexts for a given plaintext where data is taken from $D$ different keys and $N = K$. This approach could also be applied to a multi-key system using counter mode making use of a tradeoff such as (4) which targets the key and unpredictable initial count. In this case, $D_{iv}$ represents data from different key/IV combinations and the constraint $D_{iv} \leq V$ need not be applied. Further, applying a distributed approach resulting in a tradeoff of the form of (12) may make some systems vulnerable.

As an example, consider counter mode of AES-128, where $K = 2^{128}$ and $V = 2^{32}$. Assume that we have collected data from $2^{12}$ IVs from each of $2^{12}$ different keys, so that $D_{iv} = 2^{24}$. Enlisting $2^{20}$ processors in the TMTO attack, we can find one of the used keys with an online complexity of $T_0 = M_0 = 2^{70.7}$ and a preprocessing complexity of $P_0 = 2^{116}$, where the restriction $D_{iv}^2 W \leq T_0$ holds as required for the distributed HS approach.

## 6 Numerical Results for Some Tradeoffs

In this section, we highlight a few cases to illustrate the applicability of the distributed TMTO attack. The data presented considers two key sizes of 80 bits (Table 1) and 128 bits (Table 2). A key size of 80 bits is consistent with the typical use of a lightweight block

**Table 1:** TMTO Results $T_0/P_0$ for 80-bit Keys

| $K = 2^{80}$ | DEKS | $V = 1$ | $V = 2^{20}$ | $V = 2^{40}$ |
|---|---|---|---|---|
| $W = 1$, $D_{iv} = 1$ | $2^{80}$ | $2^{53.3}$ / $2^{80}$ | $2^{66.7}$ / $2^{100}$ | $2^{80}$ / $2^{120}$ |
| $W = 1$, $D_{iv} = 2^{10}$ | $2^{80}$ | $2^{53.3}$ / $2^{80}$ | $2^{60}$ / $2^{90}$ | $2^{73.3}$ / $2^{110}$ |
| $W = 2^{20}$, $D_{iv} = 1$ | $2^{60}$ | $2^{33.3}$ / $2^{60}$ | $2^{46.7}$ / $2^{80}$ | $2^{60}$ / $2^{100}$ |
| $W = 2^{20}$, $D_{iv} = 2^{10}$ | $2^{60}$ | $2^{33.3}$ / $2^{60}$ | $2^{40}$ / $2^{70}$ | $2^{53.3}$ / $2^{90}$ |

**Table 2:** TMTO Results $T_0/P_0$ for 128-bit Keys

| $K = 2^{128}$ | DEKS | $V = 1$ | $V = 2^{32}$ | $V = 2^{64}$ |
|---|---|---|---|---|
| $W = 1$, $D_{iv} = 1$ | $2^{128}$ | $2^{85.3}$ / $2^{128}$ | $2^{106.7}$ / $2^{160}$ | $2^{128}$ / $2^{192}$ |
| $W = 1$, $D_{iv} = 2^{20}$ | $2^{128}$ | $2^{85.3}$ / $2^{128}$ | $2^{93.3}$ / $2^{140}$ | $2^{114.7}$ / $2^{172}$ |
| $W = 2^{20}$, $D_{iv} = 1$ | $2^{108}$ | $2^{65.3}$ / $2^{108}$ | $2^{86.7}$ / $2^{140}$ | $2^{108}$ / $2^{172}$ |
| $W = 2^{20}$, $D_{iv} = 2^{20}$ | $2^{108}$ | $2^{65.3}$ / $2^{108}$ | $2^{73.3}$ / $2^{120}$ | $2^{94.7}$ / $2^{152}$ |

or stream cipher, while the 128-bit key could represent an application that uses AES-128. The results in the tables could represent a tradeoff attack using the DK approach of a single-key system based on counter mode or a stream cipher. The table values assume equal complexity for the online time and memory, i.e., $T_0 = M_0$. The tradeoff expression of (12) is applied and the constraints $D_{iv} \leq V$ and $W \leq T_0$ are satisfied. For $V > 1$, $P_0 = KV/(D_{iv}W)$ resulting in

$$T_0 = \frac{P_0^{2/3}}{W^{1/3}} \tag{21}$$

which can be used to derive the values in the tables. However, for the case of $V = 1$ (that is, a predictable initial count in counter mode or a stream cipher with no IV), data cannot be used in the tradeoff and $P_0 = KV/W$ with (21) still suitable.

For both key sizes, various IV sizes are given and the complexity presented for cases of differing amounts of data, $D_{iv}$, and number of processors, $W$. For reference, the appropriate distributed exhaustive key search complexity (DEKS) is also presented for each case. Each TMTO case given in the tables has the online time complexity and the preprocessing complexity for an individual processor presented in the format $T_0/P_0$.

It is obvious from the tables that there are many scenarios in which distributed TMTO attacks could be made more effective than a distributed exhaustive key search. Most notably, if $V = 1$, one Hellman table can be constructed straightforwardly to cover just the keys. In this case, although the use of data from multiple IVs is not applicable, applying a distributed approach can result in extremely small online time complexities - as low as $2^{33.3}$ for a lightweight cipher with an 80-bit key using $2^{20}$ processors. For cases with $V > 1$, using data drawn from a modest number of IVs can result in a compromise of the security of the cipher. For example, with $K = 2^{128}$ and $V = 2^{32}$, using data from only $2^{20}$ IVs and applying $2^{20}$ processors results in a TMTO attack with an online time complexity of $2^{73.3}$ and a preprocessing time complexity of $2^{120}$. Hence, the online time complexity is substantially better than the distributed exhaustive key search complexity of $2^{108}$, while the preprocessing complexity is only slightly worse.

# 7 Conclusions

In this paper, we have discussed the characterization of distributed TMTO attacks on ciphers. Not surprisingly, distributing Hellman's approach can be highly effective, scaling both time and memory by the number of processors. Other tradeoff approaches such as the rainbow table method and the BG method are not as well suited to a distributed

approach. The BS method benefits from a distributed approach in both time and memory, but the benefit of data in the tradeoff is not scaled by the number of processors involved. We have also described the application of distributed tradeoff attacks in relation to stream ciphers and have shown that TMTO and distributed TMTO approaches can be applied to counter mode in scenarios where the entropy of the initial count is too small. In particular, distributed TMTO attacks are of concern in the context of lightweight cryptography, where key sizes are smaller and the cryptanalytic gain of distributing the attacks could seriously compromise the security of some systems.

## References

[1] Martin E. Hellman. A cryptanalytic time-memory trade-off. *IEEE Trans. Information Theory*, 26(4):401–406, 1980.

[2] S. Babbage. A space/time tradeoff in exhaustive search attacks on stream ciphers. In *European Convention on Security and Detection, IEE Conference Publication No. 408*, pages 161–166, 1995.

[3] Jovan Dj. Golić. Cryptanalysis of alleged A5 stream cipher. In Walter Fumy, editor, *Advances in Cryptology - EUROCRYPT '97, International Conference on the Theory and Application of Cryptographic Techniques, Konstanz, Germany, May 11-15, 1997, Proceeding*, volume 1233 of *Lecture Notes in Computer Science*, pages 239–255. Springer, 1997.

[4] Alex Biryukov and Adi Shamir. Cryptanalytic time/memory/data tradeoffs for stream ciphers. In Tatsuaki Okamoto, editor, *Advances in Cryptology - ASIACRYPT 2000, 6th International Conference on the Theory and Application of Cryptology and Information Security, Kyoto, Japan, December 3-7, 2000, Proceedings*, volume 1976 of *Lecture Notes in Computer Science*, pages 1–13. Springer, 2000.

[5] Jin Hong and Palash Sarkar. New applications of time memory data tradeoffs. In Bimal K. Roy, editor, *Advances in Cryptology - ASIACRYPT 2005, 11th International Conference on the Theory and Application of Cryptology and Information Security, Chennai, India, December 4-8, 2005, Proceedings*, volume 3788 of *Lecture Notes in Computer Science*, pages 353–372. Springer, 2005.

[6] Dorothy E. Denning. *Cryptography and Data Security.* Addison-Wesley, 1982.

[7] Philippe Oechslin. Making a faster cryptanalytic time-memory trade-off. In Dan Boneh, editor, *Advances in Cryptology - CRYPTO 2003, 23rd Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 2003, Proceedings*, volume 2729 of *Lecture Notes in Computer Science*, pages 617–630. Springer, 2003.

[8] Orr Dunkelman and Nathan Keller. Treatment of the initial value in time-memory-data tradeoff attacks on stream ciphers. *Inf. Process. Lett.*, 107(5):133–137, 2008.

[9] Gildas Avoine, Pascal Junod, and Philippe Oechslin. Characterization and improvement of time-memory trade-off based on perfect tables. *ACM Trans. Inf. Syst. Secur.*, 11(4):17:1–17:22, 2008.

[10] Jin Hong and Sunghwan Moon. A comparison of cryptanalytic tradeoff algorithms. *J. Cryptology*, 26(4):559–637, 2013.

[11] Fabian van den Broek and Erik Poll. A comparison of time-memory trade-off attacks on stream ciphers. In Amr Youssef, Abderrahmane Nitaj, and Aboul Ella Hassanien, editors, *Progress in Cryptology - AFRICACRYPT 2013, 6th International Conference*

*on Cryptology in Africa, Cairo, Egypt, June 22-24, 2013. Proceedings*, volume 7918 of *Lecture Notes in Computer Science*, pages 406–423. Springer, 2013.

[12] Johan Borst, Bart Preneel, and Joos Vandewalle. On the time-memory tradeoff between exhaustive key search and table precomputation. In *Proceedings of the 19th Symposium in Information Theory in the Benelux, WIC*, pages 111–118, 1998.

[13] Jin Hong, Ga Won Lee, and Daegun Ma. Analysis of the parallel distinguished point tradeoff. In Daniel J. Bernstein and Sanjit Chatterjee, editors, *Progress in Cryptology - INDOCRYPT 2011 - 12th International Conference on Cryptology in India, Chennai, India, December 11-14, 2011. Proceedings*, volume 7107 of *Lecture Notes in Computer Science*, pages 161–180. Springer, 2011.

[14] Jung Woo Kim, Jungjoo Seo, Jin Hong, Kunsoo Park, and Sung-Ryul Kim. High-speed parallel implementations of the rainbow method based on perfect tables in a heterogeneous system. *Softw., Pract. Exper.*, 45(6):837–855, 2015.

[15] Gildas Avoine, Xavier Carpent, Barbara Kordy, and Florent Tardif. How to handle rainbow tables with external memory. In Josef Pieprzyk and Suriadi Suriadi, editors, *Information Security and Privacy - 22nd Australasian Conference, ACISP 2017, Auckland, New Zealand, July 3-5, 2017, Proceedings, Part I*, volume 10342 of *Lecture Notes in Computer Science*, pages 306–323. Springer, 2017.

[16] Ga Won Lee and Jin Hong. Comparison of perfect table cryptanalytic tradeoff algorithms. *Des. Codes Cryptography*, 80(3):473–523, 2016.

[17] Martin Hell, Thomas Johansson, Alexander Maximov, and Willi Meier. The Grain family of stream ciphers. In Matthew J. B. Robshaw and Olivier Billet, editors, *New Stream Cipher Designs - The eSTREAM Finalists*, volume 4986 of *Lecture Notes in Computer Science*, pages 179–190. Springer, 2008.

[18] National Institute of Standards and Technology. *NIST Special Publication 800-38A: Recommendation for Block Cipher Modes of Operation*, December 2001. crsc.nist.gov/publications/detail/sp/800-38a/final.

[19] Alex Biryukov, Sourav Mukhopadhyay, and Palash Sarkar. Improved time-memory trade-offs with multiple data. In Bart Preneel and Stafford E. Tavares, editors, *Selected Areas in Cryptography, 12th International Workshop, SAC 2005, Kingston, ON, Canada, August 11-12, 2005, Revised Selected Papers*, volume 3897 of *Lecture Notes in Computer Science*, pages 110–127. Springer, 2005.

# Appendix A: Summary of Tradeoffs

Table 3 contains a summary of all tradeoffs discussed and applied in this paper. Tradeoff expressions and preprocessing complexity, as well as target applications and meaningful restrictions on tradeoff parameters, are presented.

**Table 3:** Summary of Tradeoffs

| | Tradeoff | Preprocessing | Target Applications and Restrictions |
|---|---|---|---|
| Exhaustive Key Search | $T = K$, $M = 1$ | $P = 0$ | block cipher key<br>stream cipher key |
| Full Dictionary Attack | $T = 1$, $M = K$ | $P = K$ | block cipher key<br>stream cipher key |
| Hellman | $TM^2 = K^2$ | $P = K$ | block cipher key |
| BG | $TM = N$ | $P = N/D$ | stream cipher state<br>$D = T$ |
| BS | $TM^2D^2 = N^2$ | $P = N/D$ | stream cipher state<br>$D^2 \leq T$ |
| HS | $TM^2D_{iv}^2 = (KV)^2$ | $P = KV/D_{iv}$ | stream cipher key/IV<br>counter mode key/IV<br>$D_{iv}^2 \leq T$ |
| DK | $TM^2D_{iv}^2 = (KV)^2$ | $P = KV/D_{iv}$ | stream cipher key<br>counter mode key<br>$D_{iv} \leq V$ for single-key |
| Distributed Exh Key Srch | $T_0 = K/W$, $M_0 = 1$ | $P_0 = 0$ | block cipher key<br>stream cipher key |
| Distributed Full Dict Att | $T_0 = 1$, $M_0 = K/W$ | $P_0 = K/W$ | block cipher key<br>stream cipher key |
| Distributed Hellman | $T_0M_0^2W^3 = K^2$ | $P_0 = K/W$ | block cipher key<br>$W \leq T_0$ |
| Distributed BG | $T_0M_0W = N$ | $P_0 = N/(DW)$ | stream cipher state<br>$D = T_0$ |
| Distributed BS | $T_0M_0^2D^2W^3 = N^2$ | $P_0 = N/(DW)$ | stream cipher state<br>$D^2W \leq T_0$ |
| Distributed HS | $T_0M_0^2D_{iv}^2W^3 = (KV)^2$ | $P_0 = KV/(D_{iv}W)$ | stream cipher key/IV<br>counter mode key/IV<br>$D_{iv}^2W \leq T_0$<br>$D_{iv} \leq V$ for single-key |
| Distributed DK | $T_0M_0^2D_{iv}^2W^3 = (KV)^2$ | $P_0 = KV/(D_{iv}W)$ | stream cipher key<br>counter mode key<br>$W \leq T_0$<br>$D_{iv} \leq V$ for single-key |

# Appendix B: Distributed Attacks

We now briefly discuss exhaustive key search and dictionary attacks in the context of a distributed attack. These are attacks at the two extreme ends of the time-memory tradeoff spectrum. We present them as they provide a benchmark with which to analyze the effectiveness of distributed TMTO attacks.

For a block cipher, exhaustive key search works by considering 2 or 3 known inputs for

which the outputs are intercepted from the system under attack. Encryptions are done for all possible candidate keys and the resulting outputs are checked to see which ones are consistent with the intercepted outputs. For a stream cipher with a known IV, for each candidate key, a sufficient number of keystream bits are produced and compared to the bits of the keystream derived from a known plaintext/ciphertext pair from the system under attack, with the correct key presumed to be the candidate key which is successful in the comparison. It is obvious $T = K$, $M = 1$ and, since there is no preprocessing, $P = 0$.

Exhaustive key search can be easily distributed by simply dividing up the search space between a number of participating processors. With $W$ processors operating concurrently, the time complexity for an individual processor is $T_0 = K/W$ with the amount of memory being negligible, implying we can let $M_0 = 1$. Since all processors are operating concurrently, in fact, this also represents the total time taken in the attack. No preprocessing is required, meaning $P_0 = 0$.

In contrast, a full dictionary attack can be applied to a block cipher by storing in memory all possible keys and outputs which correspond to each key, assuming a given input value. (Here, we use the adjective "full" to indicate that all keys have outputs stored.) The table should be sorted according to output values. When the given input and its corresponding output is acquired from the system under attack, the output is looked up in the table and the corresponding key is taken as correct. The output could correspond to some spurious keys, as well as the correct key, so some filtering using a couple of other input/output pairs may be necessary. A similar attack can be structured for a stream cipher.

For the full dictionary attack, the online time complexity represents the time to look up in the sorted table, which is strictly logarithmic in the table size, but since we shall assume that logarithmic factors are negligible (as is presumed in TMTO attacks), $T = 1$. The memory requirement involves storing data for all keys and, hence, $M = K$, while the preprocessing time involves filling in the table and thus $P = K$. For a block cipher in ECB mode, assuming a chosen plaintext attack, the data requirement is negligible for the attack.

Consider now the distributed approach to the full dictionary attack. This can be achieved by involving $W$ processors to each prepare $K/W$ of the total entries in the table. The total amount of memory is still $M = K$, but, for each individual processor, the amount of memory is $M_0 = K/W$ and the preprocessing time for each processor is $P_0 = K/W$, which is the same as the overall preprocessing time assuming that all processors concurrently prepare their tables. The online time is still a small fixed value (assuming that any necessary communication time is small), and we can assume that $T_0 = 1$. For a sufficiently large $W$, in some cases, a previously impractical memory requirement, $M$, can be made practical as $M_0$ for the individual processors.

Note that, by necessity, distributed attacks require elements of inter-processor communication that do not exist in non-distributed implementations. However, invariably the cost of such communication can be quite small and, at worst, proportional to the number of processors. For example, to implement a distributed exhaustive key search, each processor must be given a range of keys to search and be given the appropriate data values to work on. This could be done by communication from one central controlling processor to each processor individually and, hence, is directly proportional to the number of processors. In our analyses of distributed attacks, we assume that the communication costs are negligible and far outweighed by the benefit of reduction in time complexity and memory requirements. Admittedly, if $W$ is chosen to be large and, for attacks which use data, if $D$ is very large, these communication costs may become prohibitively large.