Rediscovery of Time Memory Tradeoffs

Jin Hong[1] and Palash Sarkar[2]

[1] National Security Research Institute
161 Gajeong-dong, Yuseong-gu, Daejeon, 305-350, Korea
jinhong@etri.re.kr
[2] Cryptology Research Group, Applied Statistics Unit,
Indian Statistical Institute, 203, B.T. Road, Kolkata, India 700108
palash@isical.ac.in

Abstract. Some of the existing time memory tradeoff attacks (TMTO) on specific systems can be reinterpreted as methods for inverting general oneway functions. We apply these methods back to specific systems in ways not considered before. This provides the following startling results. No streamcipher can provide security equal to its key length; some important blockcipher modes of operations are vulnerable to TMTO; and no hash function can provide preimage resistance equal to its digest length.

Keywords: time memory tradeoff

1 Introduction

Consider the following situation. Suppose, you know that in some future time, you will be given a problem chosen from a fixed public problem set, the answer to which belongs to a fixed public answer set. The problems in the problem set cannot be solved directly, but given any element of the answer set, it is easy to identify which problem it is an answer to.

There are two extreme ways one can deal with this situation. When you are given the problem, you could try every possible choice until you find the correct answer. This would be time consuming. Another method would be to do this time consuming operation before the problem is given, at your leisure, storing the answer-problem pairs in a table. You will be able produce the correct answer just by looking in the table, when the specific target problem is given, but this is at the cost of pre-computation time and memory for storing the table.

This situation arises frequently in cryptanalysis. For example, you know the encryption algorithm (problem set) beforehand and can prepare for the day when a specific target ciphertext (problem) is given. Time memory tradeoff attack(s), henceforth referred to as TMTO, is a method for coming somewhere between the two extreme solutions.

There has been a variety of TMTO applied on specific cryptographic algorithms. Sometimes the time consuming preparation is actually practical on some real world systems. For example, using around \$12,000, one may prepare a table for DES, with which keys may be recovered under half an hour using a single

$12 FPGA device [20]. Also, long-term use of an algorithm could allow for such table preparation.

In this work, after reviewing some of these methods, we first point out that many of these specific methods are applicable to situations that are more general than it was originally developed for. That is, we reinterpret TMTO on specific systems as methods for inverting general oneway functions.

The second contribution of this paper is application of these generalized TMTO back onto various specific systems in ways that were not tried before. This will produce the following startling results, which are true to various degrees, depending on the environment of the system in use.

1. In view of TMTO, no streamcipher can provide security level equal to its key size. In particular, huge internal state does not guarantee TMTO resistance.
2. Using cipher block chaining with random IVs does not foil TMTO on blockciphers. It is possible to utilize multiple data with blockcipher TMTO. Some important blockcipher modes of operations are vulnerable to TMTO.
3. The preimage resistance of a hash function cannot reach the security level provided by its digest length.

All of these conclusions run contrary to popular beliefs of the crypto community.

The last contribution of this paper is to describe the new TMTO application methods in a uniform way so that they may later be applied to other systems in a similar manner.

2 Review of TMTO

TMTO consists of two phases. In the *pre-computation* phase, a table related to the system in consideration is constructed. Throughout this paper, P will denote the time needed for this pre-computation, and M will denote the size of memory needed in constructing and storing this table. During the *online* phase, an explicit target is given and the attacker is asked to return an element related to this target from some search space of size N. The time taken for this reply will be denoted by T.

Complexity of the attack is usually taken to be the sum or maximum of T and M. Hence, for a meaningful TMTO, both T and M should be at least smaller than N. It is customary not to take the pre-computation time P as adding to the attack complexity. In other words, the attacker is given unlimited amount of time in preparation. We shall now quickly go over some of the TMTO theory. We ask the readers to consult the original papers for a better understanding of the algorithms.

2.1 Hellman

A TMTO on DES was given by Hellman [13]. He assumes a chosen plaintext attack situation and the objective of the attacker is to recover the secret key, given a single ciphertext which is known to be an encryption of the chosen plaintext. The search space will be the set of all (56-bit) keys and hence is of size $N = 2^{56}$.

Pre-computation Attacker fixes a (64-bit) plaintext $\mathbf{p}$ which is likely to be encrypted by the victim in the near future. A string of 8 ASCII space characters would be a good choice. Consider the function f that sends an arbitrary key k to an encryption of $\mathbf{p}$ under the key k.

$$f : k \mapsto \mathrm{DES}_k(\mathbf{p}). \tag{1}$$

This is a function which is easy to compute, but hard to invert.

Next, any[3] function h that maps 64-bit values to 56-bit values is chosen. Just dropping the last 8 bits would be one of the simplest such examples. Define $g = h \circ f$ to be the composition of f and h so that it sends 56-bit values again to 56-bit values. The stage is now set for table construction.

Starting from a random key $k = k_{0,0}$ the function g is applied iteratively t-many times, where t is to be determined later. The intermediate values are denoted by

$$k_{0,i+1} := g(k_{0,i}). \tag{2}$$

The starting point and ending point pair $(k_{0,0}, k_{0,t})$ is recorded in a table and another random key $k_{1,0}$ is chosen as a starting point. This process is repeated until m pairs of key points $\{(k_{i,0}, k_{i,t})\}_{i=0}^{m-1}$ are collected in the table. The table should be sorted according to the second component of the pairs. The undetermined values t and m are chosen to satisfy the condition

$$t^2 m = N, \tag{3}$$

called the matrix stopping rule. With help from the birthday paradox, one can show that this insures not too many of the $k_{i,j}$s overlap.

The table of m entries just created covers about $(t \cdot m)$-many keys. Attacker starts over with a different choice of h and constructs another table. This process is repeated until t-many tables are prepared, and hence all $t^2 m = N$ keys are covered. The choice of a different h keeps the number of keys covered by more than one table to a minimum.

Online phase During the online phase, a single ciphertext $\mathbf{c}$, which is known to be an encryption of our chosen $\mathbf{p}$, is provided. Attacker checks if the 56-bit value $h(\mathbf{c})$ appears as the second component of any of the entries in the t tables. Iteratively application of g to $h(\mathbf{c})$ is tried until a match

$$g^x(h(\mathbf{c})) = (g \circ \cdots \circ g)(h(\mathbf{c})) = k_{i,t} \tag{4}$$

is found from some table. We can check that

$$\begin{aligned} (g^x \circ h)(\mathbf{c}) = k_{i,t} &= g^t(k_{i,0}) && (5)\\ &= (g^x \circ g \circ g^{t-x-1})(k_{i,0}) && (6)\\ &= (g^x \circ h)(f(g^{t-x-1}(k_{i,0})), && (7) \end{aligned}$$

[3] Some choices are better than others, but we shall not concern ourselves with such details.

and hence there is a high chance that

$$\mathbf{c} = f(g^{t-x-1}(k_{i,0})) = \mathrm{DES}_{g^{t-x-1}(k_{i,0})}(\mathbf{p}). \tag{8}$$

In other words, the given target ciphertext $\mathbf{c}$ is an encryption of the chosen plaintext $\mathbf{p}$ under the key $\mathbf{k} = g^{t-x-1}(k_{i,0})$. Thus the attacker has recovered the secret key of the victim.

TMTO curve To store the t tables, each containing m key pair entries, memory of size $M = tm$ is required. The pre-computation time taken for table preparation is $P = N$, since we have covered all N keys. We have disregarded the less significant constant factor corresponding to each entry size and also the time taken for table sorting. During the online phase, g is applied at most t times, and this is searched for in t tables, so the online computational complexity is $T = t^2$, disregarding the less significant table lookup time factor.

With these, recalling (3), one may state the TMTO curve

$$TM^2 = N^2 \quad \text{and} \quad P = N. \tag{9}$$

Conversely, given any T and M satisfying this equation, one can find appropriate t and m which allows us to proceed with the described algorithm, recovering the secret key in time and memory complexity T and M.

One particular point on this curve is

$$T = M = N^{\frac{2}{3}}. \tag{10}$$

This shows one may recover the DES key in online time shorter than key exhaustive search time under chosen plaintext attack scenario.

The TMTO algorithm described for DES can be applied to other blockciphers with minimal change. It suffices to choose h to send binary strings of block size to strings of key size. Hence, under TMTO, given an arbitrary blockcipher with key space of size N, one may always recover the key in time and memory complexity of $N^{\frac{2}{3}}$. Every blockcipher is vulnerable to TMTO.

It is known that curve (9) reduces to

$$TM = N \quad \text{and} \quad P = N \tag{11}$$

with a better typical point of

$$T = M = N^{\frac{1}{2}}, \tag{12}$$

if the oneway function $g = h \circ f$ is a single cycle permutation. That is, if it is a permutation and its iterative application runs though all N elements.

2.2 Fiat and Naor

Even though Hellman formulated his TMTO mostly as an attack on blockciphers, he had stated that his method may be applied in inverting any oneway function. More explicitly, TMTO on general oneway functions shall refer to the following scenario.

1. The attacker is first given a oneway function $f : X \to Y$.
2. Some time P for pre-computation is allowed, during which the attacker prepares a table of size M.
3. Then, given a target point $y \in Y$, utilizing his table, the attacker finds $x \in X$ satisfying $f(x) = y$ within time T.

In the blockcipher situation, one considered the oneway function mapping a random key to the encryption of a fixed chosen plaintext. The attacker aims to invert this function, given a single point in its image space.

A more careful analysis of Hellman's *any function* claim was given in [8], arguing that it is possible to construct explicit oneway functions which resist Hellman's method. An algorithm for inverting functions that follows the TMTO curve

$$TM^3 = N^3 \quad \text{and} \quad P = N \tag{13}$$

was also given. This is said to be applicable to any function. One particular point on this curve is

$$T = M = N^{\frac{3}{4}}. \tag{14}$$

This is weaker than (10) and (12), but is very useful, considering the fact that it works for *arbitrary* oneway functions.

2.3 Babbage and Golić

Streamciphers first received TMTO through independent works by Babbage [5] and Golić [10]. Let the internal state of the streamcipher in consideration allow for N different states. Given D different keystreams of length $\log N$, the attacker aims to recover one of the internal states corresponding to any one of the keystreams. Once a state is found, the streamcipher may be run forward to decrypt the following ciphertext.

Pre-computation Define the function f to send an internal state of $\log N$ bits to a keystream of $\log N$ bit length.

$$f : (\log N)\text{-bit state} \mapsto (\log N)\text{-bit keystream}. \tag{15}$$

Create a single table containing $(s, f(s))$ pairs, choosing each state s at random. The table should contain $M = N/D$ entries and be sorted according to the second component.

Online phase Going over each of the D keystreams in turn, search the table for a pair containing it as its second component. Since $D \cdot M = N$, the birthday paradox states that we have a good chance of finding one. If a match is found, the corresponding first component is the internal state the attacker was looking for.

TMTO curve We have already stated that the memory requirement for this TMTO is $M = N/D$. It suffices to look for entries in the single table D times, so the time complexity is $T = D$. Hence, ignoring some of the data, if needed, one can see that any point on the curve

$$TM = N \quad \text{with} \quad P = M \tag{16}$$

gives an attack, as long as $1 \leq T \leq D$. For example,

$$T = M = D = N^{\frac{1}{2}} \tag{17}$$

constitutes an attack.

In fact, this suggests a design principle for streamciphers and Babbage [5] states that

> if a secret key length of k bits is required, a state size of at least $2k$ bits is desirable.

Similarly, Golić [10] states that

> a simple way of increasing the security level ...... is to make the internal memory size larger.

2.4 Biryukov and Shamir

Another development was seen in [6]. The works of Hellman and Babbage-Golić, described above, was combined to bring a new TMTO on streamciphers. As with the work of Babbage and Golić, objective of the attacker is to recover any one of the many internal states of the streamcipher, given D-many different keystreams.

Pre-computation Assume that the internal state can take N different values and define function f as in (15). Choosing random permutations to take place of h, define $g = h \circ f$ and create multiple tables as in Hellman's TMTO. As before, each table contains m entries and covers tm states. We still abide by the matrix stopping condition (3), but only t/D tables are created, covering only $tm \cdot t/D = N/D$ of the N states.

Online phase Let $\mathbf{k}$ be any one of the D keystreams given. Iteratively apply g to $h(\mathbf{k})$ until the $(\log N)$-bit value $h(\mathbf{k})$ matches the second component of some the entry in the t/D tables. The tables cover N/D states and D data points are tried, so there is a good chance of finding one match.

Once a match is found, the state leading to the keystream is recovered as in Hellman's method and the streamcipher may be run forward from that position onward.

TMTO curve The (t/D)-many tables, each of size m, require a storage space of size $M = tm/D$. Looking through each of the t/D tables, each taking time t, and repeating for each of the D data points costs time complexity $T = t^2$. This leads us to the TMTO curve of

$$TM^2D^2 = N^2 \quad \text{and} \quad P = N/D, \tag{18}$$

valid when $1 \leq D^2 \leq T$. For example,

$$T = M = N^{\frac{1}{2}} \quad \text{with} \quad D = N^{\frac{1}{4}} \tag{19}$$

constitutes an attack. Time and memory complexity of this point is equal to that of (17), but the need for data is smaller. This strengthened the view that the internal state of a streamcipher should consist of at least twice the number of bits used for key.

3 Unified view of TMTO

The work of Fiat-Naor certainly targeted the inversion of a *general* oneway function. Once the algorithm is understood, it becomes obvious that Hellman's method can also be applied to arbitrary oneway functions. Even though it is not clear as to exactly which class of functions Hellman's argument is valid for, the method itself can be tried on arbitrary oneway functions. In practice, for most cryptographic applications, Hellman's method seems to work well enough.

Though less obvious if confronted separately, the works of Babbage, Golić, and Biryukov-Shamir are also general methods of inverting oneway functions.

1. The attacker is given a streamcipher algorithm and the oneway function f mapping the internal state to a keystream of certain length is considered.
2. Time is given to the attacker for table preparation.

Only the last step, the final objective of the attacker, is slightly different from the notion of TMTO as a general oneway function inverter introduced in Section 2.2.

3. Given D-many target points $y_i \in Y$, the attacker recovers any one of the $x_i \in X$ satisfying $f(x_i) = y_i$.

Hellman could not use multiple data points, because in his chosen plaintext scenario, the input was a single fixed secret key, and hence, apparently, there was only one corresponding data point. With hindsight, we may now say that Hellman could also have used larger data size and produced the TMTO curve (18), had he considered the situation where recovering just one of the D keys that encrypted the fixed plaintext was the attacker's objective.

We do not intend to say that every TMTO developed for application to a specific system can be used to invert arbitrary oneway functions. The well known baby-step giant-step algorithm for solving DLP is one counterexample. Seen in the general setting, it inverts the oneway function $x \mapsto g^x$, but utilizes the algebraic structure of this mapping and hence is not applicable to general oneway functions.

Each TMTO is a way of inverting a certain specific oneway function. Many of them, in particular, the ones mentioned in the previous section, can be applied to *general* oneway functions. In the following sections, we shall see how these generalization of specific methods can be applied back to other specific situations. Application to streamciphers and blockcipher modes of operation will be the most interesting.

It is still unclear as to which class of oneway functions each TMTO is successful on with high probability. The ones mentioned in the previous section tend to be more successful the closer they are to a single cycle permutation. Actually, as the next lemma shows, we can expect a random orbit of a randomly chosen permutation to draw a cycle that covers about half the space it acts on, so it suffices for the oneway function to be close to a permutation.

Lemma 1. *The cycle length expectation value of a random orbit in a randomly chosen permutation on N elements is $\frac{N+1}{2}$.*

Proof. Let us fix an element x in an arbitrary set of size N. Consider the probability P_k that the orbit length of x by a random permutation is k. We should start by counting all permutations that include x in an orbit of length k.

To write down one such permutation, we can first list the k elements contained in the x-orbit following the order they appear in the orbit starting with x itself, and then record how the rest of $(N-k)$ elements are mixed among themselves. Hence there are

$$\left(\binom{N-1}{k-1}\cdot(k-1)!\right)\cdot(N-k)! = (N-1)!$$

many permutations containing x in an orbit of length k.

This shows $P_k = (N-1)!/N! = 1/N$ and the expectation value of cycle length is

$$E = \sum_{k=1}^{N} k\cdot P_k = \sum_{k=1}^{N}\frac{k}{N} = \frac{N+1}{2}.$$

Thus we can expect the cycle length of a random orbit of a random permutation to be about half of the maximum reachable.

There are many TMTO on specific systems that we have not considered. In particular, the ideas of distinguished points attributed to Rivest [7] and rainbow tables [17] are direct extensions of the TMTO discussed in this section and can be used in the general setting.

One idea we have not investigated is looking for TMTO on specific systems that cannot be generalized to arbitrary oneway functions, but can be applied to some *family* of oneway functions. For example, careful look at some specific TMTO could reveal it applicable to all oneway group homomorphisms.

4 Streamcipher

We shall apply previous TMTOs, reinterpreted as general oneway function inverters, to streamciphers. The point of application will be different from those that were tried by the original TMTOs. A suitable oneway function will be identified, some argument supporting possibility of access to multiple data will be given, and we shall see its consequences. At the end of the section we shall discuss implications of this on the design of streamciphers.

4.1 The simple case

Let us be given a streamcipher algorithm that takes a k-bit key. Our search space is the key space of size $N = 2^k$. Consider the following oneway function f.

1. Take as input a single k-bit key.
2. Place the key into the state as described by the cipher algorithm. For example, several copies of it might be padded into the state register.
3. Go through the initialization process as described by the cipher algorithm. Most ciphers will require running it for several clocks without producing output.
4. Take the first k bits of keystream as output for the function f.

We shall refer to the first starting bits, as a *prefix* of the keystream corresponding to the key. For a well designed streamcipher, this function sending k-bit key to k-bit prefix will be very close to a permutation. For otherwise, it should be possible to devise some distinguishing attack.

Next, let us argue that utilizing multiple data when applying TMTO to this oneway function is reasonable. Consider the situation of a dummy terminal session. Assume that each session is encrypted with a new key, and that the first encrypted text of a session is the (fixed) login screen so that the keystream prefix of each session is always exposed. Each session we observe gives one data point.

We have already seen that our oneway function is similar to a permutation. Suppose, for the moment, that it is close enough for TMTO curve (18) to be applicable. Referring to the specific point (19), we may say that if we observe $D = 2^{\frac{k}{4}}$ sessions, we can recover the key for *one* of these keystreams in time and memory complexity $T = M = 2^{\frac{k}{2}}$. Once the key is obtained, the corresponding whole session can be decrypted and the attacker is satisfied with having read the session of a random victim. We have taken the TMTO which was originally applied to the internal state of a streamcipher and have applied it to the key space. No modification of the TMTO algorithm was necessary. Only a slight change of view was needed.

The large data requirement $D = 2^{\frac{k}{4}}$ could be realistic for a large computing center, but if you believe this is totally unrealistic, one can use some other suitable point from (18), or in the extreme case, fall back to curve point (10), which uses only one data point, and say that the security of the streamcipher is at most $T = M = 2^{\frac{2}{3}k}$. If you are even more sceptical and are of the opinion that f is very different from a permutation, you can use curve point (14), which is successful on arbitrary functions, and say that the security level is at most $2^{\frac{3}{4}k}$. In any case, this shows that no streamcipher can provide security level equal to its key size. [4]

At this point, we must confide that the above approach of applying TMTO to the key space is not a new idea. After (re-)discovering this attack, we found the following statement in Hellman's work [13].

> . . . , but the same approach works with a synchronous stream cipher. The first k bits of keystream are taken as the $f(K)$ function, where k is the number of bits of key.

We could have waved this away, saying that at the time of his writing, the key size was usually equal to state size, and being written in this form was pure coincidence. However, hiding in the appendix of a more recent paper that broke WEP [9] is the following statement, which more definitely mentions the situation just considered.

> In this case, the pairs in the database are [secret key, prefix of the output stream], and the attack requires prefixes from a large number of streams (instead of a single long stream).

Through the works of Babbage [5], Golić [10], and Biryukov-Shamir [6], we have all learned that a big internal state size is a necessary condition for a streamcipher to resist TMTO. Over the last few years, this has led streamcipher designers to incorporate huge internal states. What is interesting is that designers have started to believe that huge state size is a sufficient condition for TMTO resistance and have forgotten and neglected the above known results. This slip can even be seen in [10] itself.

> . . . doubling the memory size, from 64 to 128 bits, is very likely to push the attacks beyond the current technological limits. Note that the secret session key size need not be increased to 128 bits.

Also, most recent streamcipher proposals have erroneously quoted their huge memory size as indications of TMTO resistance. However, we have seen that, under the framework of TMTO, *no streamcipher can provide security level equal to its key length.*

4.2 Streamciphers with IV

The situation with streamciphers have changed somewhat since the early work of Hellman, and modern ciphers now use a nonce or an initial vector (IV) in

[4] From now on, we shall not mention (14) and mainly focus on (19), giving the attacker more power. Readers can easily adjust our argument to their taste if they believe that some of the oneway functions we present are far from permutations, or if our use of multiple data seems unreasonable.

addition to the secret key. Resynchronization is more common in this situation, and obtaining large sets of data is more realistic.

Consider an environment where many short messages are encrypted, each with a different IV. Assume that the master key is seldom changed. This may happen with wireless communication frames, or maybe a disk encryption scheme where each sector is encrypted with a different IV. Assume some of these frames or sectors are known to us in the form of bare keystream. Since IVs are usually public, if we can obtain the master key to one of these frames, all other frames using the same master key would be readable.

We first need to define an appropriate oneway function. Consider the function

$$f : \{\text{master keys}\} \times \{\text{IVs}\} \to \{\text{keystream prefix}\}. \qquad (20)$$

Function f sends a random (k-bit key, v-bit IV) pair to a $(k+v)$-bit keystream prefix. So our search space is of size $N = 2^{k+v}$. For a good cipher, this mapping should be close to a bijection.

Now, recalling (19), we can see that if we have access to $D = 2^{\frac{1}{4}(k+v)}$ frames of raw prefix, we can recover a single (key, IV) pair within time and memory complexity $T = M = 2^{\frac{1}{2}(k+v)}$. In particular, if $v < k$, this complexity is smaller than key exhaustive search complexity 2^k. That is, if IV is any shorter than key, the streamcipher is vulnerable to TMTO.

Below, we state some remarks on this and give some variations to this method.

1. Putting a restriction on how many frames are encrypted before the master key is renewed does not stop this attack completely. The attacker still gets to know one of the many master keys.
2. Making the state initialization process more complex has completely no effect on this TMTO. It does not matter whether the initialization process is linear or nonlinear, and it does not matter whether the key can be recovered from state.
3. The known part of keystream need not be at the very beginning. As long as they are fixed positions in the keystream, they don't even need to be continuous. The oneway function can be defined to match the known part.
4. If IV is XORed into the key before being placed into the internal state, we could set the domain of the oneway function to be at that position. In general, the domain of f should be at the point of least entropy occurring during the initialization process.
5. Using IVs in a predictable manner, as in the case of counter supplied IVs, effectively reduces the IV space, making TMTO even easier. Fixed IV situation was actually dealt with in the previous subsection.
6. Readers with full understanding of actual TMTO algorithms will find that fixing IV for each table has the effect of reducing table lookup count, hence making it more efficient. However, this has only a small effect on the overall time complexity. Note that this is closely related to defining your oneway function as

$$(k\text{-bit key}, \text{IV } V) \mapsto (k\text{-bit keystream prefix}, V).$$

7. Pre-computation time $P = N/D = 2^{\frac{3}{4}(k+v)}$ will usually be larger than exhaustive key search 2^k, but this fact is immaterial in the framework of TMTO, where unlimited time is usually allowed for pre-computation.

4.3 Design principle for streamciphers

If one views TMTO as a threat to streamciphers, one of the following measures should be taken.

1. Ensure that in every implementation of your cipher, the collective entropy of key and IV will always be at least twice of that provided by your key size. In particular, your IV length should be at least equal to key size and it should not be used in a predictable way. During your state initialization process, the collective entropy of key and IV should not be allowed to decrease below twice key size.
2. If you are designing a general purpose streamcipher, and don't know in what manner your cipher is going to be used, claim security level corresponding to half your key size. Then, arbitrary use of IV may be allowed. Entropy of internal state after initialization should not be smaller than provided by key size.

Since streamcipher designs would be aiming for the most demanding environments, we realize that the above guidelines are not very realistic. Related comment, especially in connection with the recent ECRYPT call for streamcipher proposals [3], can be found in Appendix A.

5 Blockcipher modes of operation

Hellman has shown that arbitrary blockcipher is vulnerable to TMTO under the chosen plaintext attack scenario. In this section, we apply TMTO to most of the blockcipher modes of operation available today. We were able to do this successfully on every mode we have considered. This seems to indicate that, in general, *all blockcipher modes of operation are vulnerable to TMTO.*

5.1 ECB, CTR, OFB

In Hellman's TMTO on DES, the key size was shorter than block size. Since we want the oneway function to be similar to a permutation, when key size is greater than block size, his method may not work too well. But such cases may be dealt with by working with a multi-block plaintext. Once the key is obtained, all other ciphertext may be decrypted. We have thus argued that ECB mode is vulnerable to TMTO under chosen plaintext attack scenario, if the plaintext is chosen to be as long as the key.

Counter mode is in a very similar situation if counter update is predictable. The counter value predicted to be used in the near future gives us a basis for the chosen plaintext attack, and when the corresponding ciphertext is given, the key may be recovered in time shorter than key exhaustive search. After this, all other ciphertext may be decrypted.

The OFB mode of operation is essentially a streamcipher with IV, and arguments of the previous section apply. That is, under the known plaintext scenario, if key length is any longer than block or IV length, OFB is vulnerable to TMTO when access to enough known plaintext is given.

5.2 CBC, CFB, TBC

The situation with CBC may seem a bit different from the above at first. There is widespread belief that

1. it is not possible to use multiple data in blockcipher TMTO, and that
2. cipher block chaining with random IVs will foil TMTO on blockciphers.

The following quote from [19] gives some evidence that the first of these is currently common sense.

> Generic time/memory tradeoff attacks on stream ciphers ($TM^2D^2 = N^2$) are stronger than the corresponding attacks on block ciphers ($TM^2 = N^2$) since they can exploit the availability of a lot of data.

Root of second claim seems to lie in the concluding section of Hellman's original paper [13], where remarks on known and chosen plaintext situations are mixed in a way that invites misunderstanding. We shall now look at the CBC mode of operation to argue that neither of these beliefs are true. Actually, we have already commented in Section 3 that the first is not strictly true.

To ease explanation, we shall assume a key size which is twice the block or IV size. For example, 256-bit key for AES. We work in the chosen plaintext attack model, and fix a plaintext of 3 block size (384 bits for the above example). Under key k and IV v, the CBC mode of operation sends a 3-block message $\mathbf{m} = m_1||m_2||m_3$ to $c_1||c_2||c_3$, where

$$c_1 = E_k(m_1 \oplus v), \quad c_2 = E_k(m_2 \oplus c_1), \quad \text{and} \quad c_3 = E_k(m_3 \oplus c_2). \tag{21}$$

Here, E_k denotes single block encryption under key k. Define our oneway function f to send an arbitrary 3-block value $k||v$ to the 3-block ciphertext $c_1||c_2||c_3$, obtained by encrypting a fixed 3-block chosen plaintext $\mathbf{m} = m_1||m_2||m_3$. Our search space size is $N = 2^{|k|+|v|} = 2^{3b}$, where b denotes block size. With a single 3-block ciphertext data which is an encryption of the chosen $\mathbf{m}$, TMTO point (10) applies with complexity $2^{2b} = 2^{|k|}$ and we fail to obtain a meaningful attack.

So we should find some way to incorporate multiple data into the picture. There are at least the following four ways to do this. These can even be combined for even more data.

1. As commented in Section 3, assume that our fixed chosen plaintext was encrypted multiple times, each with a distinct key (and IV). Let the attacker's

objective be the recovery of any one of them. An example would be a multi-user environment with the attacker satisfied if he can harm any one of the users.
2. Assume that the chosen plaintext was encrypted multiple times using the same key but under different IVs.
3. We may assume that our chosen plaintext appeared several times in a single long session. This is very similar to the second situation.
4. An even more plausible solution is to choose your 3-block plaintext to consist of 3 identical blocks. That is, we look for encryptions of $\mathbf{m} = m||m||m$. We then assume that our ciphertext data contains an encryption of a series of $(D+2)$-many m. This would give us D many encryptions of $\mathbf{m}$ in the following manner: $f(k||c_i) = c_{i+1}||c_{i+2}||c_{i+3}$, where $i = 0, \ldots, D-1$, and $c_0 = v$. Inverting f on $c_{i+1}||c_{i+2}||c_{i+3}$ will provide k (and c_i).
   For example, if we are dealing with a 64-bit blockcipher, we let m be 8 ASCII space characters. Then an encryption of a single row of space characters (80 of them) would contain 8 encryptions of $\mathbf{m}$.

Thus, we have incorporated multiple data into TMTO on CBC mode of operation. With enough data points, TMTO curve point of (19) applies. The attack complexity is $2^{\frac{3b}{2}} = 2^{\frac{3}{4}|k|}$, which is smaller than key exhaustive search. If IVs are public, recovering the key from any one of these session would allow other sessions to be decrypted. Even if IVs are not public, we could just ignore the first block and think of the second block as starting a CBC mode with the IV set to the first ciphertext block, decrypting from the second block on.

In more general terms, under the assumption that TMTO curve point (19) applies, CBC mode of operation is vulnerable to TMTO if key length is any longer than IV length. As with the situation in streamciphers, predictable use of IVs works to the attacker's advantage.

The situation with CFB is exactly the same with CBC. It suffices to exchange (21) with

$$c_1 = m_1 \oplus E_k(v), \quad c_2 = m_2 \oplus E_k(c_1), \quad \text{and} \quad c_3 = m_3 \oplus E_k(c_2). \tag{22}$$

Also, same argument applies to tweakable blockciphers [16] running in TBC mode. It suffices to use tweak in place of IV.

5.3 OCB, OMAC

The mode OCB [18] produces MAC in addition to the ciphertext. Encryption part of OCB is similar to ECB, except that one extra key-like element is used for each block of encryption. These key-like elements are derived from a key and nonce pair, and is updated for each block of additional encryption.

From the view point of TMTO, the MAC output part is no different from the ciphertext. As before, we use the chosen plaintext attack scenario and define the oneway function to send $(\text{key}, \text{nonce})$ pair to $(\text{ciphertext}||\text{MAC})$.

Because of the key-like element that changes with each block, only the first two of the four ways to obtain multiple data mentioned for CBC can be used.

The amount of data available determines how effective this TMTO would be. If (19) applies, OCB is vulnerable to TMTO whenever key is longer than nonce. At the worst, with a single data point, curve point (10) is applicable and OCB is vulnerable to TMTO if key is over twice as big as nonce. As with streamciphers, using nonce predictably adds to the attacker's advantage.

OMAC [14] is a NIST standard for encryption and authentication. It is a one key CBC with the capability of producing an authentication tag. As with the case of OCB, the TMTO attack on CBC can be modified to work on OMAC.

5.4 CMC, EME

Let us consider the CMC [11] and EME [12] modes of operation. A tweak in addition to a key[5] is used. These are two-pass encryption modes and every bit of the ciphertext depends on the whole input text. Hence only the first two of the four ways to obtain multiple data mentioned for CBC can be used. TMTO should provide the attacker with a key (in addition to the tweak). This can then be used on ciphertexts using different tweaks.

We fix a plaintext and define the oneway function f as follows. The function f takes as input a pair $(\text{key}, \text{tweak})$ and encrypts the plaintext to obtain the ciphertext. This is hashed (by a collision resistant hash function) to obtain a string of length equal to $|\text{key}| + |\text{tweak}|$. This string is the output of f. In the online phase, we will have a ciphertext and can hash it to obtain a string in the range of the oneway function. Finding a pre-image of the range element will provide the secret key (and also the tweak).

These modes of operations extend a small block length pseudorandom permutation to a wide block length pseudorandom permutation. The intended application is for in-place disk encryption, where the tweak is the sector address and the plaintext block consists of the contents of the corresponding sector. Thus, block length is quite large (around 512 bytes). The reason for using hash function in the definition of the oneway function is so that we do not record this long ciphertext in the table.

It is quite possible that the contents of many of the sectors are identical, which is especially true if the sectors are not in use. In such situation, we can utilize multiple data by obtaining the ciphertexts corresponding to different sector addresses (tweaks) among the sectors containing our fixed chosen plaintext. Inverting any one of the points will reveal the master key (and the corresponding tweak), which can be used to decrypt other blocks.

5.5 Example: GSM

Our discussion so far on blockcipher modes of operation has shown that, in most cases, security level reached by a key of length longer than block length, does not corresponding to key length, in view of TMTO. For example, IDEA, KASUMI, and long key versions of AES are all subject to this problem. In this section, we

[5] Two keys are used for CMC, but we can treat them as one long key.

turn to a more specific use of blockcipher and study the use of KASUMI in GSM mobile phones. It will illustrate that the actual entropy of key and IV matters more than just their length.

The encryption algorithm for GSM mobile phones [1] is called A5/3. It is a modified version of OFB mode of operation based on KASUMI, which is a 64-bit blockcipher with key length of 128 bits. In the use of A5/3 for GSM encryption, the 128-bit key is actually a concatenation of two copies of a single 64-bit key.

The IV to be used is created by applying KASUMI to a 128-bit value with the key XORed by a fixed constant. But most part of this 128-bit value is fixed to some constant with only a 22-bit counter part updated each time a new IV is needed. Only 228 bits of keystream are used after initialization with a new IV, but this is not important for us.

We can define our oneway function as

$$(\text{64-bit key}, \text{22-bit counter value}) \mapsto \text{86-bit keystream prefix}.$$

The initialization process for creating the IV from key and 22-bit counter, and also the chaining method itself is a bit different from OFB, but this is all immaterial. It suffices for the attacker to invert the above map.

Applying the TMTO curve point (19) under the known plaintext scenario, we can recover the 64-bit key in time and memory complexity 2^{43}. This is true if $2^{21.5}$ data is available and pre-computation time $2^{64.5}$ is allowed. Since the counter used as a part of IV is only 22 bits long, it seems more reasonable to collect data that correspond to multiple master keys. In practice, this may have been obtained from multiple users. When one of these keys is recovered, it can be used to decrypt messages encrypted with the same key and different IVs.

The authors are not aware of the actual situation, but if only a small portion of the possible counter values are used in real life, i.e., if the entropy of the counter is smaller, the attacker's position is strengthened further.

6 Hash function

We shall list our findings concerning TMTO on hash functions. Relative to the streamcipher situation, it is less interesting. On the other hand, with the demand for small hash functions increasing in relation to its possible use in RFIDs, these results may have implications on hash designed for those environments.

6.1 Simple hash

We could not find reasonable application of TMTO to collision finding, especially since birthday attack already reduces the security level to half of hash length. But obtaining preimage or second preimage quickly with the added advantage of pre-computation time seem to be plausible attack scenarios not considered before.

Let us be given a hash function $h : M \to H$, where H is the message digest space. Restrict the message space and consider the oneway function

$$h : H \to H. \tag{23}$$

Any TMTO method can now invert this function. If you have multiple targets and it suffices to find preimage of any one of these targets, TMTO curve point (19) could be applicable. For digest space H of size N, time and memory complexity of TMTO is only $N^{\frac{1}{2}}$.

Even with a single target point, TMTO curve points of (10) or (14) is applicable. This shows that under TMTO, no hash function can achieve preimage resistance security level equal to its digest size.

We believe arguments of this section to be a (academically) valid attack, but in the real world, no hash of size small enough to allow the pre-computation time needed would exist.

6.2 Keyed hash and MAC

Under the chosen plaintext attack model, keyed hash (or MAC) is very similar to the ECB mode of operation. Sending key to the keyed hash value of a fixed plaintext is the oneway function to be considered. Attacker's objective is to recover the key, given the keyed hash value corresponding to the chosen plaintext. Once the key is obtained, it could be used to forge other hash (MAC) values.

Security level falls to $\frac{2}{3}$ of the key size under (10). If one believes this oneway function to be a permutation, so that (12) applies, security level falls to half of key size. If key length is larger than digest size, hash pair corresponding to two chosen plaintexts could be used to make the oneway function closer to a permutation.

Contrary to the situation with simple hash, in some real world situations, it could be tempting to use a small MAC. Suppose one sets up a session key with another party and uses it to calculate a MAC. If this MAC is always verified by the receiving party within a very short time period of session key setup, it might seem plausible to use a weak MAC. But we have shown that TMTO must be taken into account when doing this.

7 Asymmetric algorithms

In many cases of symmetric key algorithms, the security level expected of an algorithm is equal to its key size. This is far from true in the asymmetric world. Hence TMTO, the best of which only halves the security level in some sense, is less interesting here. Nevertheless, to show that TMTO is a versatile tool, we shall apply TMTO to some asymmetric algorithms in this section. None of these give an attack of complexity less than the security level each algorithm is designed for, but these could give ideas on how to apply TMTO in other (symmetric cipher) situations.

7.1 RSA encryption

Given an encryption exponent e and an RSA modulus n, consider the oneway function $m \mapsto m^e \pmod{n}$. With the pre-computation completed, when a target ciphertext is given, the corresponding plaintext can be found faster than exhaustive plaintext trial.

If some sort of padding scheme is used, the attacker is in a better position because it effectively reduces the message space. Still, this is nowhere close to the claimed security level of RSA.

7.2 ElGamal encryption

Given a public key $h = g^k$, take the encryption function $(r, m) \mapsto (g^r, h^r m)$ as the oneway function. When a ciphertext is given, the attacker recovers the corresponding random number and plaintext. If the entropy of r is a lot smaller than that of m, the online phase could be faster than exhaustive plaintext search.

This example has striking resemblance to the key+IV situation considered in previous sections and shows that even probabilistic algorithms are not completely out of reach from TMTO.

Having explained this example, we must say that it is more efficient to apply TMTO directly to the oneway function $r \mapsto g^r$ and use the recovered r to find m. As the current systems are designed to withstand this approach, our example is far from being realistic.

7.3 NTRUEncrypt

The NTRU public key cryptosystem [2] works in the ring

$$\mathcal{R} = (\mathbf{Z}/239\mathbf{Z})[x]/(x^{251} - 1).$$

That is, all elements are polynomials of degree less than 251 with coefficients taken modulo the prime 239. In its bare form, with the public key $\mathbf{h} \in \mathcal{R}$ and a binary polynomial $\mathbf{m} \in \mathcal{R}$ as the message, encryption is given by

$$\mathbf{e} = \mathbf{r} * \mathbf{h} + \mathbf{m}.$$

Here, $*$ denotes polynomial multiplication. The polynomial $\mathbf{r}$ started out as random polynomials, but to counter chosen ciphertext attacks, it is now calculated deterministically from the message $\mathbf{m}$.

So, with the public key $\mathbf{h}$ given, an attacker may consider the oneway function $\mathbf{m} \mapsto \mathbf{e}$ and prepare an associated table at his leisure. During the online phase, given a ciphertext the attacker recovers the corresponding plaintext. Since the message space is of size 2^{251}, applying (10), we see that the complexity of attack is 2^{168}.

Since an encryption is a bijective process[6], arguing that (12) might be applicable is not totally without reason. In such a case attack complexity goes down

[6] This may not be strictly true for NTRU, due to the so called wrapping failure.

to 2^{126}. But this is still larger than the best known attack on NTRU of complexity 2^{106}, which happens to be another TMTO called the meet-in-the-middle attack. As the message itself also contains some deterministic[7] padding, this number could go down a little bit further. One might also consider the situation where only short messages are encrypted or where multiple encrypted messages are given to the attacker.

7.4 Signature schemes

Many signature schemes send a triple (m, k, r) consisting of message, key, and randomizing value to a signature (x, s). Here, x is a function of the random value r and sometimes also of m, and s is a function of all inputs.

One fixes a message m likely to be signed by the victim in the near future and considers the function $(k, r) \mapsto (x, s)$. Once a table is prepared, the key k (and r) can be recovered soon after obtaining the expected signature.

Depending on the relative size of k and r, this TMTO could be efficient than key exhaustive search. However, its complexity will not go anywhere near the claimed security level of the signature schemes.

Again, as with our approach on ElGamal, we can apply TMTO to the $(r, m) \mapsto x$ part first (under chosen plaintext scenario), and use the obtained r to recover k, for a more efficient attack.

8 General framework for TMTO application

Through the arguments of previous few sections, we have seen that TMTO can be applied to many different situations in a very versatile way. In this section, let us take for granted that TMTO is a general method for inverting well-behaved oneway functions, and explain a general method for applying it to cryptographic situations.

In all of the cryptographic situations considered in this paper, under an appropriate attack scenario, we could devise a oneway function of the following form.

$$f : K \times V \to C. \tag{24}$$

Here, K denotes the secret values the attacker is trying to obtain, and V refers to the set of auxiliary values which is, in many situations, public but not controllable by the attacker. The set C contains the output values and a specific single target from this set is given to the attacker at the last stage of TMTO. What these sets refer to in the various situations considered in this paper is listed in Table 1. In some cases, V is missing from the cryptographic system, in which case we think of V as containing a single element.

[7] The message $\mathbf{m}$ also contains 80 bits of random padding, so the actual space for the real message is smaller.

Table 1. Fitting various cryptographic situations into TMTO framework

situation		K	V	C
block	ECB [13], CTR	key	-	single ciphertext block
	OFB, CBC, CFB	key	IV	ciphertext blocks
	TBC	key	nonce	ciphertext blocks
	OCB	key	nonce	ciphertext blocks + MAC
	CMC, EME	key(s)	tweak	ciphertext blocks
stream	previous [5, 6, 10]	state	-	keystream of state size
	simple	key	-	keystream of key size
	with IV	key	IV	keystream of (key+IV) size
hash	preimage	message	-	hash value
	keyed	key	-	hash value
PK Enc	RSA, NTRU	message	-	ciphertext
	ElGamal	message	randomizing value	ciphertext
signature		key	randomizing value	signature

Once a oneway function is fixed, in most cases, we will want to be able to apply f iteratively. This can be taken care of by applying a random hash

$$h : C \to K \times V. \tag{25}$$

The second thing we should consider is that most of the TMTO will apply with better success rate if $h \circ f$ is close to a bijection. As long as set C is larger than $K \times V$, for most cryptographic applications, this can be naturally expected of the system to some degree. If C is smaller than $K \times V$, one should find some way to deform f so that the image space is larger. We saw through many chosen plaintext attack scenarios and the keyed hash situation, that this could easily be done by simply increasing your plaintext length so that the output is long enough. In other situations, for example, if V contains publicly known values, using a hash $h' : V \to V'$ of appropriate length and setting

$$f' : K \times V \to C' = C \times V' \qquad (k, v) \mapsto f(k, v)||h'(v) \tag{26}$$

could be another solution. One should keep in mind that the image must correspond to some value that is either public or can be calculated from publicly available data.

We can now write up a set of guidelines for applying TMTO to a cryptographic system.

1. Identify a (oneway) function $f : K \times V \to C$, inverting which will reveal a secret information of the attacker's interest, belonging to K. This function need not be the oneway function the designer of the system had in mind.
2. K and V should be taken as small as possible, allowing it to be just big enough to reflect the actual entropy of values used.

3. If needed, adjust the function so that the entropy of function image space is equal to its input space. This will help in making the function f a bijection, hence raising the success probability of attack.
4. Lower attack complexity can be achieved if it is possible to devise an attack scenario where the attacker is given multiple target points in the image space of f and finding the inverse image of any one of those points is good enough.
5. Depending on the amount of target points available and how close function f is to a bijection, apply a suitable TMTO method to obtain a secret value in K.
6. When abundant data is at hand, TMTO curve point (19) is applicable, and the attack is meaningful whenever $|K| > |V|$. At the other extreme with one data point and oneway function of bad characteristics, we have (14), and the attack is successful when $|K| > |V|^3$.

We can summarize all this by saying that the most difficult task of applying TMTO to a cryptographic system is finding a plausible scenario of attack, preferably in which a large set of data is available. Once this is done, the rest of the process comes naturally.

9 Conclusion

TMTO is basically a oneway function inverter. Since many of the TMTO methods available today do not take advantage of any structure the system in consideration provides, they can be adopted to inverting arbitrary oneway functions.

To attack a specific system with these generic TMTO methods, it suffices to identify a suitable oneway function, inverting which will provide you with a secret. In doing this, one should open their eyes to oneway functions hiding in the system, different from the one designer of the system had in mind. Success of TMTO depends heavily on the available amount of data, so devising an appropriate scenario of attack is also crucial.

It has been known for a long time that TMTO has implications on the security of every blockcipher in ECB mode. By applying generic TMTO in ways not tried before, we have confirmed that TMTO has security implications on most blockcipher modes of operation. It would be interesting to know if there are better approaches allowing for sharing of tables between different modes.

We have also shown that TMTO affects the security of every streamcipher, not only those with small internal states.

TMTO as a general oneway function inversion technique is more powerful and versatile a tool than is currently known to this community.

References

1. 3GPP TS 55.215 V6.2.0 (2003-09), A5/3 and GEA3 Specifications. Available from `http://www.gsmworld.com`

2. Consortium for Efficient Embedded Security. Efficient embedded security standards (EESS) #1. Version 2.0, June 2003. Available from
`http://www.ceesstandards.org/`
3. ECRYPT. Call for stream cipher primitives. Version 1.2, Feb. 2004.
`http://www.ecrypt.eu.org/stream/`
4. S. Babbage, Stream ciphers — what does the industry want? *State of the Art of Stream Ciphers* workshop, Brugge, 2004.
5. S. H. Babbage, Improved exhaustive search attacks on stream ciphers. *European Convention on Security and Detection,* IEE Conference publication No. 408, pp. 161–166, IEE, 1995.
6. A. Biryukov and A. Shamir, Cryptanalytic time/memory/data tradeoffs for stream ciphers. *Asiacrypt 2000,* LNCS 1976, pp. 1–13, Springer-Verlag, 2000.
7. Denning, *Cryptogrphy and data security,* Addison-Wesley, 1982.
8. A. Fiat and M. Naor, Rigorous time/space tradeoffs for invering functions. *SIAM J. on Computing,* vol 29, no 3, pp. 790–803, SIAM, 1999.
9. S. Fluhrer, I. Mantin, and A. Shamir, Weakness in the key scheduling algorithm of RC4. *SAC 2001,* LNCS 2259, pp. 1–24, Springer-Verlag, 2001.
10. J. Dj. Golić, Cryptanalysis of alleged A5 stream cipher. *Eurocrypt'97,* LNCS 1233, pp. 239–255, Springer-Verlag, 1997.
11. S. Halevi and P. Rogaway, A tweakable enciphering mode. *Crypto 2003,* LNCS 2729, pp. 482–499, Springer-Verlag, 2003.
12. S. Halevi and P. Rogaway, A parallelizable enciphering mode. *CT-RSA 2004,* LNCS 2964, pp. 292-304, Springer-Verlag, 2004.
13. M. E. Hellman, A cryptanalytic time-memory trade-off. *IEEE Trans. on Infor. Theory,* **26** (1980), pp. 401–406.
14. T. Iwata and K. Kurosawa, OMAC: One-Key CBC MAC. *Fast Software Encryption, FSE 2003,* LNCS 2887, pp. 129–153. Springer-Verlag.
15. A. K. Lenstra and E. R. Verheul, Selecting cryptographic key sizes. *J. of Cryptology,* **14** (2001), pp. 255–293.
16. M. Liskov, R. L. Rivest, and D. Wagner, Tweakable block ciphers. *Crypto 2002,* LNCS 2442, pp. 31–46, Springer-Verlag, 2002.
17. P. Oechslin. Making a fast cryptanalytic time-memory trade-off. *Crypto 2003,* LNCS 2729, pp. 617–630, 2003.
18. P. Rogaway, M. Bellare, J. Black, and T. Krovetz, OCB: A block-ciper mode of operation for efficient authenticated ecryption. *8th ACM CCS,* ACM Press, pp. 196–205, 2001.
19. A. Shamir, Stream ciphers: Dead or alive? Presentation slides for invited talk given at *Asiacrypt 2004.* Available from `http://www.iris.re.kr/ac04/`
20. J.-J. Quisquater and F.-X. Standaert, Exhaustive key search of the DES: Updates and refinements. *SHARCS 2005.* Available from
`http://www.ruhr-uni-bochum.de/itsc/tanja/SHARCS/`

A Remarks on dealing with streamcipher TMTO

Problems with resolving TMTO Arguments of this paper has shown that TMTO is applicable to every streamcipher and that no streamcipher can provide security level equal to its key size. To steer clear of this threat, one has to either ensure that entropy of IV is at least equal to that of key, or claim security level corresponding to half of key size. Neither of these is very realistic in the actual constrained situations streamciphers are targeting.

On the other hand, as can be seen in the recent ECRYPT Call for Stream Cipher Primitives [3], there are views that resistance against TMTO should be taken as one of the minimal requirements for a streamcipher.

In this section, we attempt to persuade readers that taking into account the pre-computation time required for TMTO table creation is a more realistic way of dealing with TMTO. This is not to say that the current *rules* for TMTO that allows for unlimited preparation time is meaningless. An algorithm that allows such an attack is potentially weaker than one that does not.

Recent opinions of experts [4, 19] are that design of streamciphers should aim for the two extreme situations, namely, ultra-fast software and low footprint hardware. Whereas requiring internal state size that is twice the key size may not be too harsh for the spacious software environments, this is not the case for hardware. For example, it requires about 2K gates just to maintain a 256-bit internal state. Considering the 5K gate realization of AES, this leaves very little room for designers. New requirements for key or IV size presented by this paper is even more problematic for designs aiming for low-end hardware. The cost of communication for key agreement is so high in these environments that increasing key size is not an option.

Taking pre-computation time into account Demanding TMTO resistance for streamciphers makes it almost impossible to use it in low-end hardware environments. The only alternative seems to become more realistic and to consider just the TMTOs that use reasonable pre-computation time, waving away TMTOs that would be infeasible for the near future. The decision as to what is infeasible would be closely related to the expected lifetime of the algorithm.

Adding weight to this approach is the fact that AES is good enough for most purposes. So the use of streamciphers would be restricted to environments that are very constrained. It seems widely accepted that compared to blockciphers, streamcipher designs should lean more to the efficiency side.

As a matter of fact, it seems strange that TMTO is largely disregarded in blockcipher theory while resistance to it is demanded of with streamciphers.

Distinction between key and state TMTO There are views that TMTO on key (and IV) should be treated differently from the old TMTO on internal state. The argument is that TMTO on state is an attack on the main body of a streamcipher, whereas that on key is an attack on its specific use. A related situation is the fact that AES is vulnerable to TMTO in ECB mode, but that nobody views AES itself as weak.

Our position to this view is that the key initialization process is an essential part of a streamcipher algorithm. It is not possible to use a streamcipher without dealing with its key initialization process. Furthermore, the situations we have considered in this paper are typical uses of streamciphers. If a cipher is weak in its typical use or if it is not really possible to stop users from using it in a problematic way, security of the cipher should be reconsidered.

Conclusion On streamcipher designs that are aiming for constrained environments, one should not demand resistance to TMTOs requiring infeasible pre-computation time.

B When should we start building a table?

In this section, we consider the question of whether it makes sense to start any of your long-term exhaustive search processes today.

Moore's law It has been observed that processor power doubles every 1.5 years. Let us assume that this will be true for the foreseeable future. Going back to high school mathematics, we can write the processing power $p(t)$ at time t as

$$p(t) = \alpha \cdot 2^{\frac{2}{3}t}. \tag{27}$$

We'll take $t = 0$ to correspond to today, in which case, constant α will be our current computational power.

Example table creation Let us consider Hellmans's TMTO on AES as an example. The pre-computation stage will be an exhaustive processing of all 128-bit keys. On my desktop PC, AES encryption runs at 488 Mbps, which translates to about 2^{47}-many 128-bit blocks per year. We should consider the key schedule also. Assuming that it runs at about the same speed as the encryption, we can take

$$\alpha = 2^{46} \quad \text{“key} \mapsto \text{ciphertext” mappings/year.} \tag{28}$$

So how long would the table creating take? Solving for T in

$$\int_0^T 2^{46+\frac{2}{3}t}\, dt = 2^{128}, \tag{29}$$

we find that the table creation will end $T = 121.3$ years from now. This assumes that my computer is constantly upgraded.

Starting later What happens if we do nothing for 120 years, and only then start building the table? Our computation power will be $\alpha = 2^{46} \cdot 2^{\frac{2}{3}120} = 2^{126}$. Solving for T' in

$$\int_0^{T'} 2^{126+\frac{2}{3}t}\, dt = 2^{128}, \tag{30}$$

we find that the table creation will take $T' = 2.3$ years, hence ending 122.3 years from now. So we are late by one year than what was achievable. But, is finishing one year earlier really worth the trouble of upgrading your computer constantly for 120 years?

In general, given any computation that takes n years from now to complete, if one starts the computation n years later, it can be finished in less than 1.5 years from then on.

Discussion Notice that just one PC was assumed to be in use for this calculation. So we are not claiming that AES will be safe for 120 years. The reader is referred to [15] for a better treatment of key size choices.

Arguments of this section show that when mounting on a large exhaustive search process, it makes more sense to wait until the computing power reached allows for it to be done within a reasonable time than to start right away. The time you gain by starting today is not really worth the trouble.

In the other direction, suppose one wants to argue that a table preparation for TMTO on an algorithm is not possible for at least some n years from now. It suffices to calculate how long it would take for the table creation n years later. In short, computational power grows so fast that accumulated computation does not mean very much.