Implementation of Time Memory Trade-off attack using MPI on GPC

by

Khaled Nassar

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Applied Science
in
Electrical and Computer Engineering

Waterloo, Ontario, Canada, 2018

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

Time memory trade-off (TMTO) is a computationally intensive cryptographic attack originally introduced by Hellman in 1980. Since then many different improvements and implementations were researched and developed. In this thesis, we propose a frame work to implement TMTO with parallel computing using message passing interface (MPI) which is generic to serve a general cryptographic algorithm with flexibility. We have presented the framework development, design rationale, behavior testing, and proposal for collision handling. For the design rationale, we identified all the components, and features needed to build or expand the framework, and we identified the differences between it and original methodology proposed by Hellman. We explained the rationale behind choosing specific features for our implementation and for adding verification features like XOR cipher. We tested the behavior of the framework using mostly Simeck and partially Speck ciphers. We show that the main issue affecting the effectiveness of the generic implementation is collisions. We concluded that problem is almost completely parallel and coarse grained once we ignore the requirement for uniformly distributed random generation of the starting points. Throughout the program design and job result analysis, it became apparent that collision will be the main challenge so we proposed a fine grained collision detection and avoidance algorithm using parallel computing that should eliminate this problem and increase the coverage of the framework. The proposed algorithm relies on three layers of processes. The difference from the current approach is the middle layer that will be responsible for detecting and preventing collisions using doubly linked list structures. This will also be helpful in detecting cipher biases since it monitors the calculated block over the whole search space.

Acknowledgements

I would like to thank Prof. Guang Gong for her continuous advice and support, and opening the door for me to learn about the fascinating world of cryptography. I would like to thank Prof. Kalikinkar Mandal as well as Prof. Gong for providing the LFSR polynomials needed for this work. I would like to extend the thanks to everyone who made this thesis possible.

Dedication

This is dedicated to my beloved wife and kids and my family back home.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

The world of information security can be illustrated by using contests or competition analogy. There is the race to break into a system before it is patched, the race to patch systems before they are compromised, the competition between contesting parties to find a superior technology, the challenge to hide some information for a specific time and a challenge to try to uncover it within that time. All parties try to use the best technology they can acquire to build defensive and/or offensive scenarios. Better technology that can be viewed as superior architecture, algorithms and computation capacity decides who wins.

The world of cryptography and cryptanalysis is a part of it. Cryptographers try to develop cryptographic systems that help ensure the confidentiality of information (Encryption), and ensure the integrity of the data and the authenticity of the sender [1, p. 4-6]. Both cryptography and cryptanalysis count on intensive computation to establish and verify the security of cryptographic systems. For example, public key cryptography like RSA uses intensive computation to transport the session keys (protecting session key transport), while intruders and cryptanalysts may use brute force computation to try to break ciphers. One of the technologies that are key to increasing the performance of algorithms is parallel computing. With the resources available in today's world, parallel computing is used to increase the performance of cryptography algorithms as well as increase the performance of cryptanalysis attacks. Damrudi *et al.*[2] discussed using parallel pipeline of tree topology to increase the performance of TRSA, a public key, asymmetric algorithm. In symmetric key cryptography Fiskiran *et al.*[3] used parallel computing to increase the table lookup performance. Pramanik *et al.*[4] went further to explore DNA computing to achieve parallel cryptography.

1.1 The motivation behind the work

The motive behind the work was to explore the potentials of parallel computing by developing a framework that uses the Scinet super computer General Purpose Cluster (GPC) [5]. We intend to formalize a framework that is flexible enough to accommodate for any ciphers and potentially many attacks later on. This means that we had to develop the code ourselves. By doing so we can generate information that would help understand the effect of such parallel computing power on the race between intruders and defenders.

1.2 Our contribution

The focus of this work is to identify the components of and create a framework that utilizes a super computer MPI based technology to cater for multiple time-memory trade-off (TMTO) attacks. To do this we considered as many approaches and ideas within the TMTO domain as possible, and implemented some of them in an expandable framework that can be upgraded to implement more ideas and ciphers. The framework can be a base to test different approaches to the problem. It should also be able to compare attack times for different algorithms provided the ciphers are comparable. We can enumerate the contribution as the following:

- Framework development: We have developed a framework that is flexible enough to accommodate different ciphers and added features for the TMTO attack under MPI or message passing interface. The framework has an external part that controls versions, job runs, probing (sensitivity analysis), and result analyses. And there is the internal part which does the actual parallel computation. The internal part is programmed using C language and is denoted TMTO-MPI.

- Design rationale: We have identified components needed to build and expand the framework, and we have differentiated between our work and the original Hellman [6] approach. We explained the rationale behind our design decisions and introduced a stability verification technique using simple XOR block cipher.

- Testing: We ran tests and wrote observations about the behavior of the algorithm under different parameters and used Simeck and Speck block ciphers as out test instances. We observed and analyzed the behavior of output density, sorting, false positives and collisions. We concluded that collisions are the biggest problem that affects the effectiveness of this this approach.

- Collision handling: We proposed a fine grained parallel architecture for solving one of the most pressing issues, that is, the problem of collision detection and avoidance. This proposed approach is based on 3 layers of processes. The first layer is made of one parent process responsible for main operations like initializing variable, initiating MPI, finishing MPI, and reporting. The second layer is made of multiple custodian processes, each responsible for maintaining a doubly linked list to keep track of block values that were calculated, and to reject new block values that were previously calculated. And the third layer is made of the majority of processes and is responsible for the actual cryptographic operations.

This work is more focused on following highlighted items illustrated in Table 1.1. Other focus areas might also apply but at a much lesser weight than the highlighted primary focus area.

Table 1.1: Research primary focus area by comparisons with others

Research primary focus area by comparison with others				
Category A	Category B	Category C	Category ..	Comment
Confidentiality	Integrity	Availability		The attack scenario in this work assumes we are trying to recover a key, not forge or change information, and not disable a service.
Offensive Security	Defensive Security			This is due to the intrusive nature of cryptanalysis. Researchers try to break the security measures before a malicious intruder does.
Detective or Proactive	Preventive	Recovery		Cryptanalysis of a cipher can be categorized more towards detective or proactive countermeasure that aim to find out vulnerabilities before it is exposed in the field.

Security Product	Product Security			Since Simeck is designed to be integrated into IoT devices, it is obvious here that this is a product security issue.
Mathematical or algorithm oversight	Protocol design weakness	Implementation bug	Exhaustive search	TMTO relies on computation resources to break cipher rather than issues with mathematics, logic, design and implementation.
Computation power of attackers	Jamming and intrusion	MitM		Following Chen *et al.*[1] definition, this is an attack that depends on the computation power of attacker.
Block cipher	Stream Cipher			Simeck and Speck are block ciphers. Our implementation of XOR for framework stability verification also mimics block ciphers
Chosen ciphertext	Known plaintext	ciphertext only		TMTO is base on known plaintext scenario.
Chosen plaintext (Batch)	Chosen plaintext (adaptive)			
Encryption	Hashing	Authentication	Non repudiation	In this work we focus on retrieving encryption keys.
CBC	ECB	OFB	CTR	Since the the work focuses on one ciphertext block this is considered ECB.
PCBC		CFB		

Simeck	Speck	XOR	DES	The framework was extensively tested for Simeck. The initial discovery phase was done using DES, and the next software stabilizing phase was done using a mock XOR block ciphers. There were some Speck runs to get an initial feed back about comparing ciphers.
Software Development Security	Access Control	Operations Security	Physical Security	Within the Common Body of Knowledge for Information security as classified by (ISC)2 [7], this work obviously belongs to the Cryptography domain.
Telecommunications and Network Security	Cryptography	Legal, Regulations, Investigations and Compliance	Security Architecture and Design	
Information Security Governance and Risk Management	Business Continuity and Disaster Recovery Planning			

1.3 Organization of the thesis

Chapter 2 reviews the literature in the domain of Time-memory trade-off, and summarizes the different approaches and ideas. It also provides a chronological account of progress since the original methodology. Chapter 3 is more oriented towards the preliminaries and the building blocks that need to be understood in order the design the attack framework. Chapter 4 presents the workflow of TMTO with MPI and describes the program specifications and the rational behind it. Chapter 5 discusses the experimental runs and results.

Chapters 6 provides conclusions and future work respectively. The appendix includes an expanded version of the run analysis, and the source and shell code used to build the framework.

Chapter 2

Literature Review

2.1 On time-memory trade-off (TMTO) attacks

Hellman [6] introduced the time-memory trade-off (TMTO) approach in 1980. The general idea was trade-off the recurrent, normally unattainable, brute force or exhaustive search cost that is required to crack a cipher with a one-time table pre-computation effort. Since the storage requirements are also infeasible, Hellman was able to virtually order the lookup tables into chains each having a cryptographic relation between its rings that enables the implementor to compress the storage requirements into a fraction by only keeping the starting point and end point of each chain, and applied the method to attack DES. Since then many have worked to optimize this attack and extend it to other ciphers than DES. Oechslin [8] proposed a solution for the colliding or merging chains, and defines a set of reduction functions ($function_1, ..., function_{t-1}$). This will make it possible for chains to merge only if the collision happens at the same point t in the same table. This lead to increased performance by 7 folds compared to the original method. Standaert *et al.*[9] proposed improvement to the distinguished points based algorithm to decrease memory access and an implementation using field-programmable gate array (FPGA). Stamp [10] explains three models of time-memory trade-off (TMTO), the "population count"example, the Shank algorithm and Hellman's Cryptanalytic TMTO. The objective of TMTO is to reach an optimum point between memory usage (already populated by pre-computation) and calculation time to retrieve some result. Biryukov [11] extends the time-memory trade-off to a time-memory-key trade-off where the same plaintext is available encrypted by multiple keys. Hong *et al.*[12] proposed a variant of the distinguished points called variable distinguished points (VDP) that affected the amount to table lookups. Avoine *et al.*[13]

work focused on decreasing the overhead of the false alarms while searching for leads. Narayanan *et al.*[14] proposed a password dictionary attack that performs better than Oechslin's rainbow table attack using time-space trade-off. Biryukov *et al.*[15] extended te concept to Time/Memory/Key trade-off. Mentens *et al.*[16] were the first to implement a hardware solution based on Oechslin's rainbow table approach. Barkan *et al.*[17] formalized a general time-memory trade-off model. Ma *et al.*[18] identified a gap between Hellman's lower bound and the actual success probability. Hong [19] analyzed the cost of false positive and incorporated it into Hellman's and Oechslin's approaches.

2.2 Parallel computing approach

Karnin [20] introduced parallel computing into the mechanism and developed a time-memory-processor relation. Later Amirazizi *et al.*[21] explored the possibilities of parallel computing and it's effect on the trade-off by estimating the capital cost, solution cost and run cost. Chang *et al.*[22] proposed a new parallel algorithm for the knapsack problem that would deliver solution for n component using $O(2^{n/4})$ memory, $O(2^{n/8})$ processors, with a cost of $O(2^{5n/8})$. Li *et al.*[23] proposed a new parallel algorithm to the knapsack problem.

Sykes *et al.*[24] used MPI to implement Oechslins [8] faster time-memory trade-off attack on Windows hashes. They used SHARCNET which is the same organization that provides GPC so it is very similar to our work in terms of the homogeneous nature of the computing nodes. Their focus was to reduce the table computation time, the password cracking time, and discussing the efficiency of the parallel approach. Their paper discussed an overview of RainbowCrack, the probability of finding a match in N space in a given table defined by t, and m, and the four categories of passwords:

- Alpha characters: Letters (A to Z). It was not clear why the lower case letters (a to z) were not included as most password systems treat lower case letters as different characters.
- Alpha-numeric characters: By adding the numbers' characters from (0 to 9) to previous set.
- Alpha-numeric-symbol14: By adding 14 commonly use symbols in passwords ! @ # $ % ^ ( ) _ − = + & ∗ to previous set.
- Alpha-numeric-symbol-all: By adding ~ ‘ [ ] { } | \ : ; ” ′ < > , . ? / to the previous set. In this set space is not included but it is not common for space to be included in allowed character list for passwords.

Sykes *et al.*[24] investigated the RainbowCrack and decided that the two processes that take the most time is the table generation and sorting. This is similar to what was observed in our work. They also decided that the table generation computation can be completely done in parallel. It is not clear though if there was any assurance that random starting points do not overlap or are uniformly distributed. They defined four and worked on three different technical approaches to saving the rainbow tables to disk:

- Each process generates and writes separate files using standard native file library.
- All process write to the same files using MPI file library.
- All process communicate data to the Master process which writes to the one file. This one is the nearest to our approach.

As for sorting, Sykes *et al.*[24] broke the file into multiples and sorted each by itself using quick sort which is the same algorithm RainbowCrack uses. Then there are two other rounds using half the processors and the master processor to merge all files into one sorted file. In our work, we sorted each table alone and left it there. There was no need to combine all tables together. For searching [24] they tried two different approaches and the one that proved beneficial is to send the hash value to all processors and divide the tables amongst them and each would report its finding.

In his thesis Taber [25] used the same technique by Oechslins [8] to create rainbow tables. While Al-Khazraji [4] developed Taber's work to support more algorithms, sorting, and different operating system's passwords. The focus was still on hashing algorithms (md4, md5, sha1, lm and ntlm). When the rainbow table is designed to break password hashing the reduction function has to not only reduce the length of the key is as the case with Hellman's original work, but it needs to make sure the strings used apply to the world of passwords. In other words using bytes that cannot be used in specific password hashing algorithm should be avoided.

2.3 Implementations of different algorithms

Time-memory trade-off attack can be used to attack many ciphers. Fiat *et al.*[26] developed a rigorous TMTO that would work for any function rather that just DES as in Hellman's original work. Saarinen [27] used to attack LILI-128. Bjorstad [28] would use TMTO to attack the Grain cipher and Dinur [29] would use it to attack FX-construction based ciphers. Biryukov *et al.*[30] used the TMTO against stream ciphers. Mentens *et al.*[16]

implemented a Unix password cracker. Dunkelman *et al.*[31] implemented TMO on stream ciphers using the initialization vectors.

2.4 Attributes for measurement

Using time-memory trade-off attack to crack passwords was thoroughly discussed in [32] "A novel time-memory trade-off method for password recovery". It is concerned with using a new time-memory trade-off technique to crack password using rainbow tables. The research [32] approached the subject from the perspective of the forensic analyst, and compared against baseline of well known rainbow table methods and uses the following as criteria:

- Reduction in storage.
- Success rate.
- Collisions.

It reviews different tools available in the market that use time-memory trade-off concept introduced by [6] and optimized by [8] to attack passwords. This paper [32] first discusses how Hellman's [6] concept would apply to cracking password hashes, and it explains how the use of the reduction function might increase the collisions and decrease rate of successful recovery. The paper argues that increasing the number of tables, with different reduction function in each, would increase the success ratio while increasing the computational complexity and storage requirements. The paper [32] also reviews the concept of distinguished end points by [33] as a way to avoid loops and it's limitation and false alarms rate increase leading to a computation penalty. The new design proposed by [32] differs from the rainbow table method in that it stores the initial value with multiple outputs instead of just one.

2.5 Historical progress

Table 2.1: Historical progress

Literature Review			
Paper	Cipher type	TMTO Implementation	TMTO improvement or Results
Hellman [6] 1980	Block, DES	Hardware	Original : recover in $N2/3$ operations with $N2/3$ memory words
Rivest [33] 1982	Block, DES		Distinguished points (DP) to decrease the search time
Karnin [20] 1984			Parallel computing for knapsack problem
Airazizi [21] 1988			Parallel computing and cost estimates
Fiat *et al.*[26] 1991	Any		Generic TMTO that works for any function
Chang [22] 1994			Improved knapsack algorithm
Borst [34] 1998			Noting chain loops and merger avoidance in Distinguished points
Biryukov *et al.*[30] 2000			Stream cipher implementation and time/memory/-data trade-off
Quisquater *et al.*[35]v2002		FPGA	first FPGA implementation
Standaert *et al.*[9] 2002		FPGA	Improvement to minimize memory lookups
Saarinen [27] 2002	LILI-128		Attack LILI-128

Oechslin [8]	windows passwords hashes (LanManager)		Different reduction functions . less lookups, less mergers, no loops
Li *et al.*[23] 2004			New algorithm for parallel knapsack problem
Avoine *et al.*[13] 2005	Unix passwords		False alarm detection using chekpoints
Biryukov *et al.*[15] 2005			Time-memory-key trade-off
Narayanan *et al.*[14] 2005			Password dictionary attack that performs better than Oechslin
Barkan *et al.*[17] 2006			Formalized a general model
Mentens [16] 2006	Unix password	FPGA	First hardware design based on Oechslin's rainbow table
Hong *et al.*[12] 2008		FPGA	Variable distinguished points (VDP)
Avoine *et al.*[36] 2008			Characteristic enabled measurement
Dunkelman *et al.*[31] 2008	Stream Ciphers		TMTO based on known Initialization vectors of stream ciphers
Taber [25] 2008	LM MD5 SHA1	Parallel computing	Parallel computing is worth the effort for TMTO attack
Ma *et al.*[18] 2009	Reduced AES-128		More accurate success probability for Hellman's approach
Hong [19] 2010	AES-128	Simulation	Analyze the cost of false alarms.

Bjorstad [28] 2013	Grain v1 (used in eSTREAM, the ECRYPT Stream Cipher Project)	N/A	Showed that there exist attacks against Grain but with disputable relevance due huge complexity of precomputation
Sykes *et al.*[24]	MS-Windows hashes	Parallel computing MPI - rainbow tables	Improved the table generation time while using MPI.
Dinur [29] 2015	FX-construction (PRINCE, PRIDE)		Successful attack with remedy proposal
Al-Khazraji [4] 2015	NTLM passwords	Parallel computing using MPI	Extended Taper's work to support more algorithms and features. Also decreased the rainbow table generation time.

To summarize, we can see that since Hellman's original proposal the Time-Memory tradeoff development over the last 38 years went in different direction:

- Advancements in the algorithm to increase coverage, for example by developing the tables.
- Advancements in the algorithm to increase response time, for example by enhancing sorting.
- Advancements in the algorithm to decrease false positives, for example by collision avoidance using distinguished points.
- Supporting different types of cryptographic families like block ciphers, stream ciphers and hashing.

- Supporting different ciphers.
- Trying different software and hardware implementation, both linear and parallel.

Chapter 3

Preliminaries

3.1 Time-memory trade-off

As described in [33] Brute force (exhaustive search) and precomputed table lookup are two simple ways of attacking ciphers. The time-memory trade-off is a combination of the two approaches. Borst *et al.*[34] also presents exhaustive search and table pre-computation as two ends of a spectrum and focuses on minimizing memory access while applying the TMTO. Hong *et al.*[37] reinterpret TMTO as a way to invert one way functions. They discuss different approaches for TMTO and develop an unified view of it.

3.1.1 Brute force and dictionary attacks

In the world of communication and computer security, brute force or exhaustive search means that the attacker will try all possible permutation of a solution to find the one that solves the problem. For example, if the attackers wants to find a 16 bits key that was use to encrypt a message, they would try 2^{16} or 65536 possible permutation to decrypt the ciphertext and then apply some extra logic to filter for valid answers. Another example in the case of on-line password cracking, where the attackers would try all permutation of permitted characters in an authentication system and try to login to that system. If the login succeeds then that was the right password. The permitted characters could be upper case alphabet, lower case alphabet, numbers, special character or a combination of two or more of these sets. Smarter attacks would take into consideration characters that are not allowed to be used in a specific system as it will significantly decrease the time

complexity. Brute force attack will always succeed with enough time and computing power to finish all permutations. This is not always available for the attacker. On the other end of this spectrum are smarter, lower complexity but not guaranteed to succeed attacks like dictionary attacks which depend on trying variations of highly probably passwords rather than trying all permutations. A third example would be the off-line password attack where it is given that the attackers were able to obtain a list of the hashed passwords. The brute force or dictionary attacks in this scenario would be to try the different password and verify if the match the hash for a specific account. When time-memory trade-off attacks are applied to password systems, special care must be taken to convert the ciphertext outputted by an encryption operation into usable string as demonstrated by Taber's [25] demo cipher system.

3.1.2 Pre-computed tables

In a known plaintext attack where the plaintext is PT, if the attackers have two column array of keys and ciphertexts pairs that result from encrypting the same plaintext PT with all possible keys, then finding the key is just a lookup problem. If the array is sorted based on the ciphertext, then the lookup operation complexity is very small. In reality, the storage requirement is not attainable.

3.1.3 The super case of brute force and pre-computed tables

In 1980 [6] introduced the concept of time-memory trade-off. This attack demonstrated that the attacker can keep T number of different key results in one entry by reusing the output ciphertext as a key to re-encrypt the known plaintext again. This meant that there exists a cryptographic relation between all the intermediate states of this chain. Hellman's algorithm [6] is as follows:

- Table pre-computation:
  - An array of M words is randomly generated to be uniformly distributed in the space of possible key values. This has the complexity of $\mathrm{O}(M)$ and is saved in memory as M elements representing starting points.
  - Each one of those array elements is encrypted and reduced T times. The result is also saved in memory representing end point. This has the complexity of $\mathrm{O}(M * T)$ which is equivalent to $\mathrm{O}(N)$ since $N = T * M$ for cipher of key values N.

  - Sorting: array is sorted based on end points to reduce the complexity of the search. This would vary according to the used sorting algorithm but it could have a worst case $O(N^2)$.

- Searching has a complexity of $O(N^{(2/3)})$: All end points are searched for the ciphertext but the complexity can be ignored due to the sorting done in previous step. If a match is found we calculate the key by starting at the starting point and regenerate the chain up to T-1. If there is no match, the ciphertext is used to encrypt the plaintext and the array is searched again. If a match is found this time then the chain is recalculated up to t-2 this time and so on. This will have the complexity of $O(t * m)$ which is different from $O(T * M)$ or $O(N)$ because there is no significant gain from increasing m or t beyond $mt^2 = N$.

3.2 Simeck

Simeck was introduced in [38] and it is a lightweight block cipher family that is designed based on SIMON and SPECK. The authors [38] compare the performance of the the three ciphers and give the specification of the cipher focusing on the hardware implementation. Since applying the time-memory trade-off attack using Message Passing Interface (MPI)[39] on General Purpose Cluster (GPC)[5] requires software implementation of the protocol, we will use the code available at [40] as foundation to create the attack. Simeck specific cipher are written in the form Simeck$2n/mn$ where n is the word size in bits which makes the block size $2n$, and m is how many words make the key[38]. The available versions of Simeck (Simeck32/64, Simeck48/96, Simeck64/128) have the key size equal double the block size. The large key size is used to facilitate the key scheduling functionality of the cipher. While each of the encryption rounds uses only one word of the key, the key rotation function rotates the key works so all of them get used [40]. In our demonstration of this attack we will use m=2 to make the key size the same as the block size. Simeck Depends on a round function:

$$R_{k_i}(l_i, r_i) = (r_i \oplus f(l_i) \oplus k_i, l_i),$$
$$f(x) = (x \odot (x \lll 5)) \oplus (x \lll 1)$$

where "$\lll$ "represents the left rotation operation. The round function is used for both the encryption, and the key scheduling. It take two words of size n (2n) and copies the left most significant word l_i into the right least significant word r_i. Cyclic shift to the left by 5 is applied to l_i and then bitwise-ANDed to the original l_i. This is again XORed to the

original l_i that has been cyclically shifted by 1 to the left. The result is XORed with the key word k_i and XOR'd again with r_i If we expand the R function:

$$r_{i+1} = l_i,$$
$$l_{i+1} = r_i \oplus ((l_i \odot (l_i \lll 5)) \oplus (l_i \lll 1)) \oplus k_i.$$

Using the same logic to revert the round function R and create Reverse Round R^{-1}:

$$l_{i-1} = r_i,$$
$$r_{i-1} = l_i \oplus ((r_i \odot (r_i \ggg 5)) \oplus (r_i \ggg 1)) \oplus k_{i-1}.$$

Simeck is a Feistel cipher structured on a nonlinear feedback shift register (NLFSR) with input similar to DES [38]. Key scheduling uses the same primitives to create the keys for each round.

3.3 Speck

Simon and Speck ciphers, according to Beaulieu *et al.*[41] were designed to be generalist block cipher that can adapt with the changing and unforeseen requirements of the world of IoT. Though other solution exists that are simple or small Simon and Speck aim is to provide a simple and small security solution. The researchers discuss some of these solution and how AES implementations fail to meet those two criteria together. Speck is very similar to Simeck as Simeck was based on the same design ideas as Speck and Simon. The same operation are used: Bitwise AND, bitwise XOR, modular addition and subtraction, left and right circular shifts. The main two functions are the key scheduling and the encrypt/decrypt. This family of ciphers is designed to cater to different requirements depending on the system constrains and it was scrutinized by the cryptographic community [41]. In our comparisons we will use the implementation provided by [42].

3.4 Parallel computing and MPI

Parallel computing enables a computer program to be split into multiple jobs each running on a different processor. While there are many was to achieve parallel computing this work uses Message Passing Interface (MPI) on the General Purpose Cluster (GPC) super computer [5] provided by Canada Compute's Scinet and hosted at the university of Toronto. GPC has 30240 cores organized into nodes of 8 cores with 16GB of memory per node [5].

Submitting jobs is limited to 48 hours of walltime for 32 cores. More cores means booking smaller walltime. Figure 3.1 shows the default total computing power allowed to submit on GPC. The numbers were calculated based on information in [5]. An important thing to take into consideration is that booking books whole nodes.

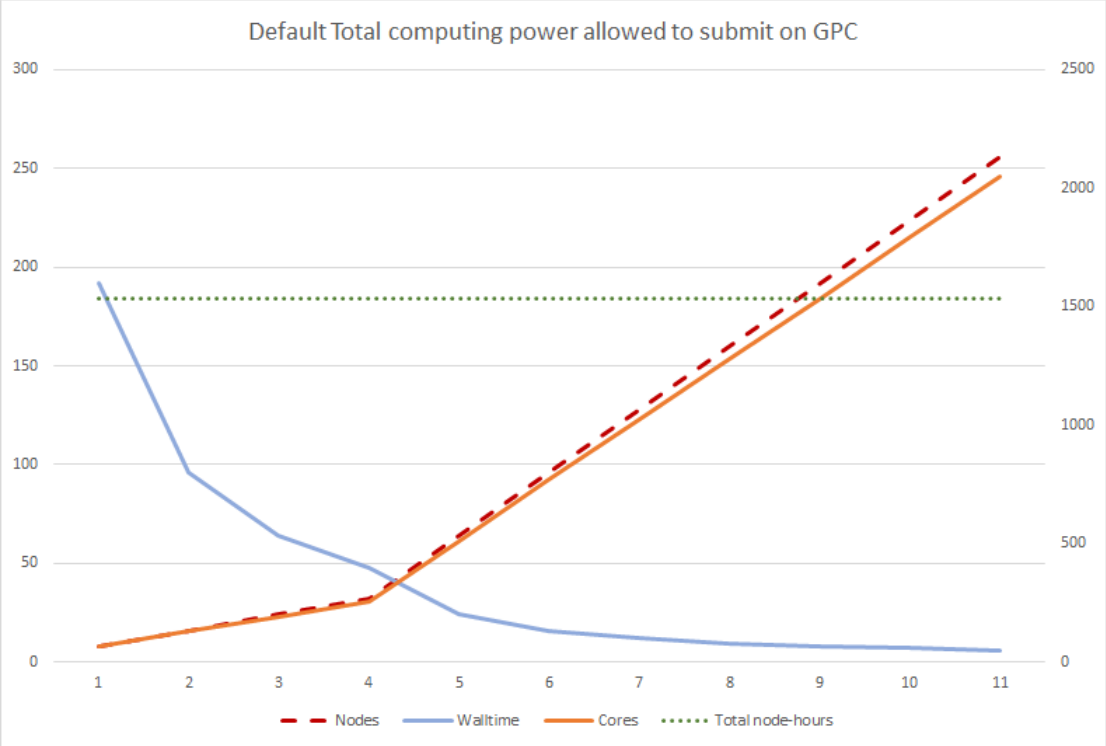

Figure 3.1: Default total computing power allowed to submit on GPC

Figure 3.1 plots nodes and walltime on the left axis and plots cores and total node-hour on the right axis. The total node-hours is fixed while cores decrease as the walltime increases. Nodes and cores are reflections of each others as 1 node is made of 8 cores.

Chapter 4

Program Specification, Rationale and Design for TMTO in MPI

In this chapter we present our design decisions and the rationale behind them for the proposed algorithm of implementing TMTO in MPI. We use mostly Simeck, and in some jobs Speck, as instances to test the algorithm. We will first describe some of the key concepts, important variables, overall algorithms, framework structure, each phase details, and then we propose a fine-grained parallel algorithm version that is enforced with collision detection and prevention to increase the effectiveness of the algorithm, as well as gather more information about the bias of ciphers.

4.1 Key concepts

4.1.1 Reduction function

Reduction function was introduced by the first time-memory trade-off research [6] to reduce the size of the output ciphertext of a preceding encryption operation t_n. In his thesis Taber [25] defines the purpose of a reduction function as a way to map the ciphertext into plaintext. Since DES has a key size of 56 bits that is smaller than the 64 bits plaintext and ciphertext block sizes, it made sense to reduce the ciphertext by 8 bits so it can be directly reused as the key for the next encryption operation t_{n+1}. In the case of Simeck [38] and Speck [41] there might not be an urging need to do so since the key size is larger than the ciphertext and plaintext block sizes. A reduction function is still important to test scenarios where it is needed to:

- Decrease the record size to achieve larger tables with the same available storage.
- Just decrease the overall table size.
- Increase the rate possible match finding (leads).

And in all these cases the number of leads will increase and there will be more calculations to retrieve the key because the number of false leads will increase. In this work the key is already reduced since we use a modified version of the algorithms that accepts half the key size as input and duplicates it to create the internal input for the key scheduler.

4.1.2 Reduction in storage

Storage is also a key player in an actual attack scenario where tables have to be stored and retrieved. Since this work is experimental and measures many individual runs, storage was not a concern.

4.1.3 Success rate

The success rate measures how many times the search for key is successful. By using different keys to challenge the attack one ends up with multiple scenarios. Sometimes, the tables do not cover a specific key. Other times, there could be multiple successful matches.

4.1.4 Collisions

In the context of TMTO, collision refers to the fact that encrypting plaintext PT with some key $K1$ could lead to ciphertext $CT1$ and that $CT1$ could exist somewhere else in the tables. Since almost all ciphertexts get reused as keys after reduction, this means that the nodes in the two chains will be of the same value. The more collisions the more wasted storage and calculations. Ciphertext $CT1$ could exist in more than one node as a result of two encryption operations, or redundancy in generating the pseudorandom starting points, or a mix of both.

4.1.5 Distinguished points

Distinguished points we introduced by Rivest [33] to decrease the possibility of collisions. This is done by defining a set of patterns that -if reached- the chain would stop growing.

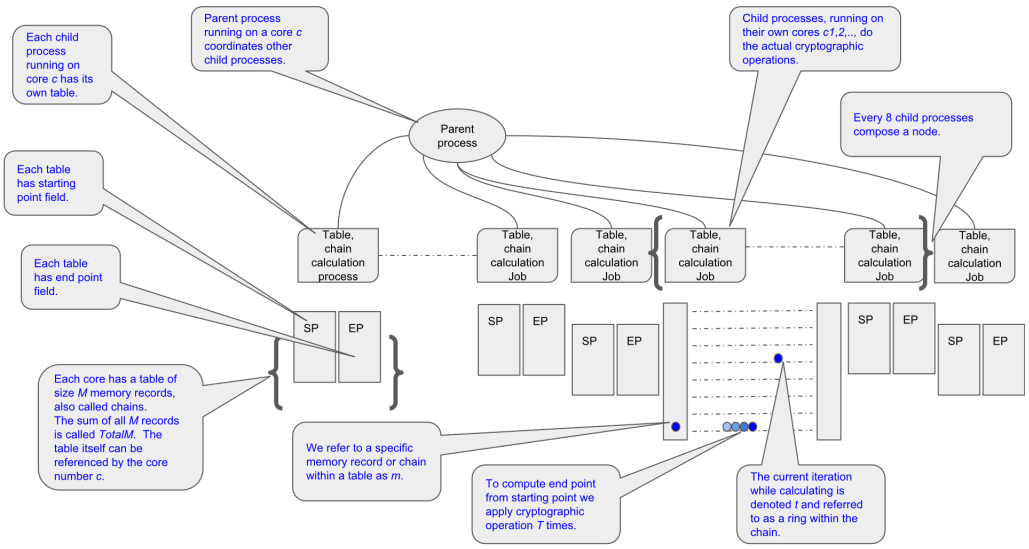

Figure 4.1: The use of main variables in the TMTO-MPI algorithm

4.1.6 Start point-end point pairs

Every chain has a randomly generated Starting point, and an end point. In this work we use LFSR to generate the starting points. We reach the end point after applying the encryption operation T times to the starting point by using the output ciphertext CT to encrypt the plaintext P over and over again. We only save the initial starting point and the resulting end point as if we virtually save the whole chain since the relation between starting point and end point is deterministic and reproducible.

4.2 Important variables

Choosing the variables or parameters for time-memory trade-off implementation was initially based on guided trial and error. This is due to the fact that initially most of the effort was to test the framework. There are some approximate bounds discussed by Saraan *et al.*[43] as they worked to define Hellman's table coverage in terms of number of distinct points. In this work we will define the important key variables in the following subsections. Figure 4.1 shows the use of the main variables within the TMTO algorithm.

4.2.1 Iterations (T)

T is the variable representing time in the time-memory trade-off. In the actual software implementation it represents the fixed number of iterations in each chain. Each starting point is used as a cryptographic key and is then used to encrypt the plaintext T times. In this research T is fixed which means that collisions cannot be avoided. The variable t refers to the current iteration the program is at while generating the chains or while searching the chains.

4.2.2 Memory (M) and ($TotalM$)

In the time-memory trade-off M is the number of memory records in the table. Each record is a structure of two words each has the same size of the block cipher's size. In this parallel implementation, there are multiple cores, and each core has it's own memory. All those M values summed together - or multiply M by $(C-1)$ - gives $TotalM$ which is the total number of memory records. Since they are guaranteed to be unique , the $TotalM$ value is important in determining coverage. The variable m refers to the current record or chain the program is working on.

4.2.3 Number of cores (C)

The number of used cores to run the test is determined by C. One core is kept to do management work, like determining quotas and reporting while the others $(C-1)$ will do the encryption, decryption, sort, search, and validation processes. Since core in the GPC infrastructure is organized in 8 cores per node structure, one can only book multiples of 8. The variable c refers to the current core that is executing some code. The $c = 0$ refers to the master process running on the first core.

4.2.4 Type of encryption (E)

E is the type of cipher (Encryption) being attacked. The framework supports four ciphers so far. Simeck and Speck were supported to be able to do some comparisons between two similar ciphers. DES was initially supported in the discovery phase of the project. Though XOR is basically a stream cipher, a simple block XOR cipher implementation was added to be used for software stability and robustness verification.

4.2.5 Indicating size difference (H)

To accommodate for ciphers that have different key-size from block-size H was introduced to indicate the difference. It was originally used for the DES cipher attack that used a reduction function. Reduction function reduces the larger block size to a smaller key size. Since with Speck and Simeck the key-size is actually larger than the block size a reduction function wasn't used and H does not indicate anything.

4.2.6 Block size (B)

Block size is defined by B. In the case of DES it is 64 bits. In Simeck and Speck it is 48 bits. In the case of XOR cipher it is very relevant because the framework can test different block sizes like 16, and 32.

4.2.7 Sorting (S)

Sorting records S based on the value of the end-point is essential to the success of time-memory trade-off. The sorting guarantees that we can retrieve the matching record very quickly using binary search. Sorting algorithms are very expensive and are assumed to be onetime cost when building the precomputed tables. However, in a test environment where we constantly try different scenarios we need to have a toggle to stop the sorting code to get faster results.

4.2.8 Notes (N)

N is the notes section that describes every run. This information can be inserted manually or can be generated by another script that tests multiple permutations.

4.2.9 Key, plaintext (PT), ciphertext (CT), challenge key (CK)

Plaintext is referred to as PT and it is fixed. Key is the the key that is used to encrypt the PT. The first key for each chain is the SP of that chain. The output of the cryptographic function is the ciphertext CT and it is used as the key to encrypt PT to generate the next node in the chain. The CK refers to the Challenge Key that is use to generate the CT that we are searching for.

4.2.10 Starting points (SP) and end points (EP)

Starting points array SP and End point array EP are the main data structure in this framework. They hold the starting point of the chains and the final results after encrypting PT T times. Originally it was one array each but because the TMTO problem is very parallel the design changed to have as many arrays for SP and EP as we have core except for the master core processor totaling $C-1$ arrays each.

4.3 Algorithm: TMTO in MPI

The algorithm consists of the table generation, search and verification, shortened as TMTO-MPI algorithm:

- Master table creation: First, there is a non parallel work done by the master process:
  - Divide the $TotalM$ over the number of cores that will be used for table generation and searching $(C-1)$. Complexity can be ignored.
  - Create M starting points using LFSR $M_{c=0,t=0,m=1}$, $M_{c=0,t=0,m=2}$, ..., $M_{c=0,t=0,m=TotalM}$. This code has complexity of $O(TotalM)$.
  - Only keep the first LFSR value for each core $M_{c=1,t=0,m=0}$, $M_{c=2,t=0,m=0}$, ..., $M_{c=C,t=0,m=0}$ and pass them to the respective cores. This code has complexity of $O(C-1)$.
- Parallel core search: Second, different cores pick up their tasks and start working on them:
  - Generate all starting points using LFSR and save them in SP array. This code has complexity of $O(M)$.
  - Generate all chains for all records in the table. This code has complexity of $O(T)$.
  - Sort using bubble sort is optional but not often used to speed up the process. This code has complexity of $O(M^2)$.
  - Search for key. This code has complexity of $O(M*T)$ because currently sorting is not enabled. This could be improved by using binary search after sorting and the M part of complexity can be ignored leading to a complexity of $O(T)$.
  - Verification by regenerating they key which has complexity of $O(T)$.
- Report to parent process: Third, all child cores processes report back to the parent process. This code has complexity of $O(C-1)$.

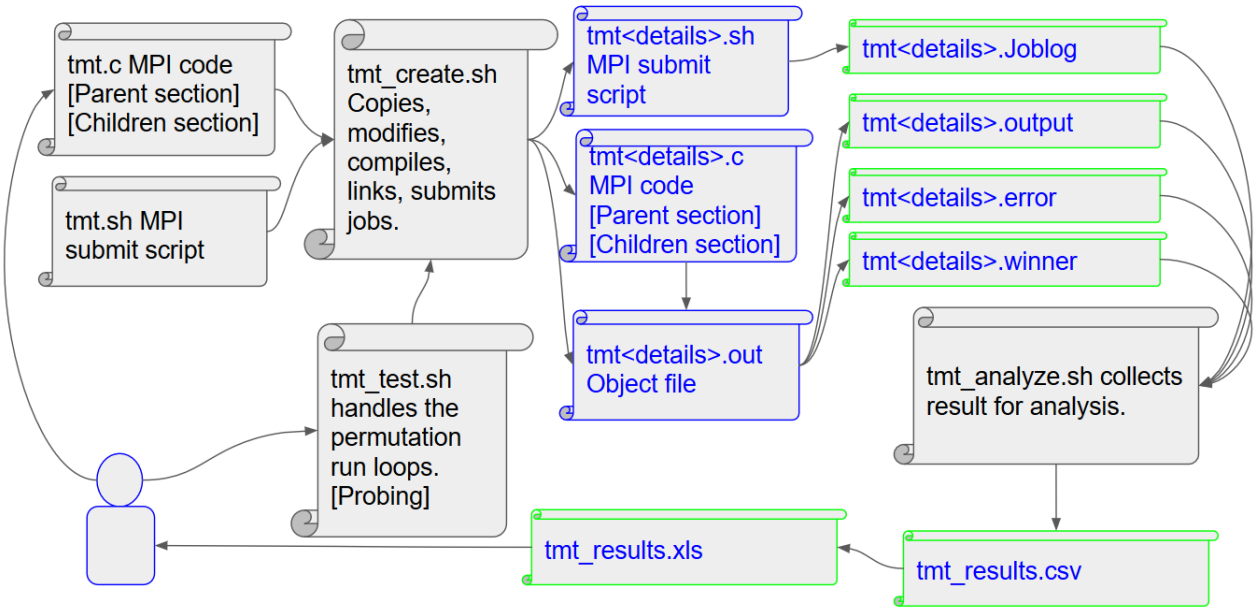

Figure 4.2: External work-flow of TMTO-MPI algorithm

4.4 Framework structure of the TMTO-MPI algorithm

The work-flow of this algorithm includes two aspects. The first is external work-flow which everything that happens before the actual object file start running on the GPC nodes, and everything that happens after the object file stops running. This includes defining the scope of the runs and all the main variables, preparing for the runs, submitting them to the queue, and afterwards collecting and analyzing the results of the runs. Figure 4.2 illustrates the external work-flow. The external framework is constructed of shell code and the researcher used some of the knowledge in [44],[45],[46],[47],[48] to solve some of the technical issues involved in writing these scripts.

The second aspect is the internal work-flow of the object file or the binary that runs on the GPC nodes after being queued by the external work-flow. It is the part that implements the TMTO-MPI algorithm. The work-flow that was used can be summarized as follows:

- Master: Create tables and initializes them.
- Master: Dispatch to child processes.

- Child: Calculate end points.
- Child: Sort records if required.
- Child: Search for the challenge CK.
- Child: Recursively re-encrypt and search.
- Child: Report back winner.
- Master: Receive and Finalize.

The main differences between this approach and the one proposed by [4] - other than the obvious different algorithms- are that the master job distribute tasks to the children do only the table calculations. It is implied that the sorting is done out of parallelization using different mechanism as well as the search later on. In our work we do include the sorting in the parallel computing by making all the children processes sort the tables they have. Since sorting is very time consuming it is disabled in most runs because the runs will search once and discard the generated tables. Figure 4.3 shows the internal structure of the program. The program is written in C and the knowledge in the following [49],[50],[51],[52],[53],[54],[55],[56],[57],[58],[59],[60],[61],[62],[63],[64],[65],[66] was consulted to solve some of the technical issue faced during the writing.

4.5 Multi-phases in the TMTO-MPI algorithm

4.5.1 Initialization

After the framework has spawned and submitted a job to the super computer, the job gets queued and then eventually started. The programs execution goes as follows:

- Defining global and other variables like M, $TotalM$, T, and C. Arrays need to be dynamically defined in the heap as it is unlimited in size if compared with the stack.
- Test the random key encryption and decryption using the requested cipher in the all processes. It outputs the test results like this for a plaintext of "DwAAgh ":

```
 [MAIN]==Random key: [0_7a_z][1_33_3][2_62_b][3_54_T][4_73_s][5_6a_j]
==CT Challenge: [0_49_I][1_5a_Z][2_26_&][3_81_~A][4_4b_K][5_9f_~_]
== CT decrypts back to: [0_44_D][1_77_w][2_41_A][3_41_A][4_67_g][5_68_h].
```

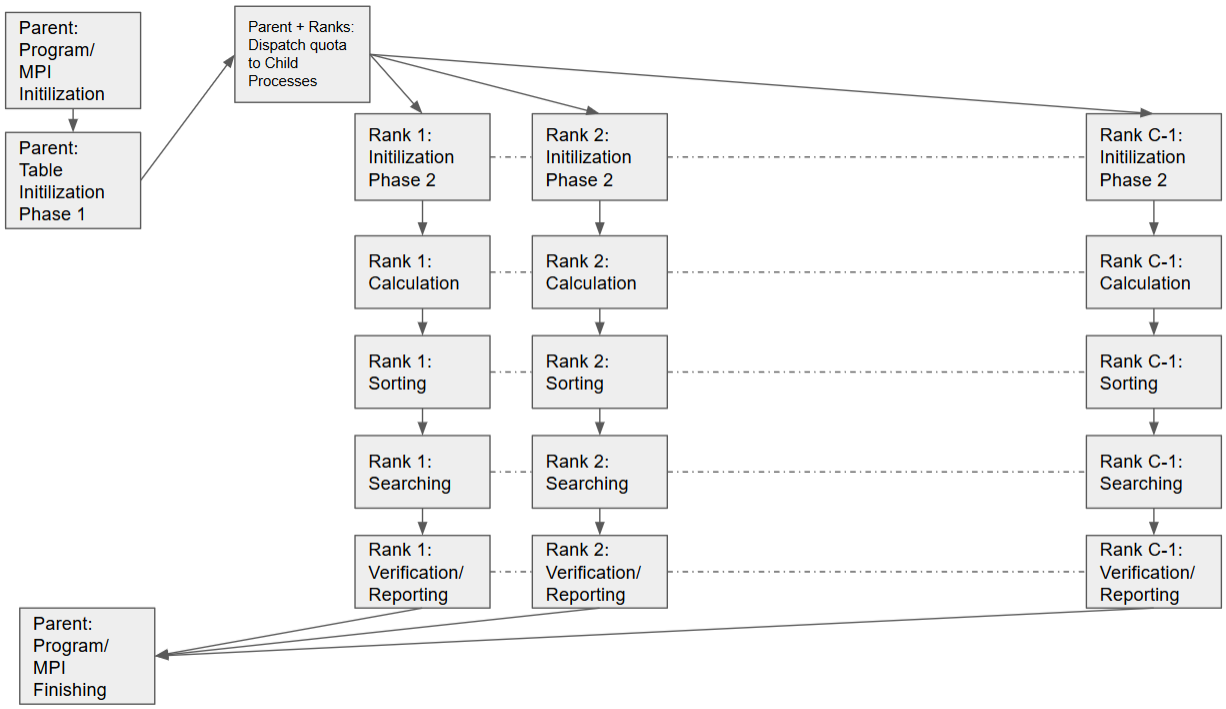

Figure 4.3: Internal structure of the program: initialization, distribution, calculation, sorting and searching in TMTO-MPI algorithm

Each byte is printed as $index, numeral format, character - format$. This is printed by all cores as it does not impose in added complexity to do so as compared to the overhead of calculating once and communicating $(C-1)$ times.

- Message Passing Interface (MPI) gets initialized at this point. Before that, each process is not aware if it is a parent or child process. Once MPI is initialized the processes communicate together and each one assumes its role as Parent or child.
- Variables that are associated with each process are defined at this point, Most importantly the starting and end point arrays of the chains in each of the processes.
- The Parent process initializes values using LFSR to be starting points for the first chain in each table. It then communicates these starting points to individual child processes.
- Each child process takes those Starting point values and use LFSR to generate all

the starting points for the tables.

Figure 4.4 shows some of the alternative features that the framework could be expanded to include.

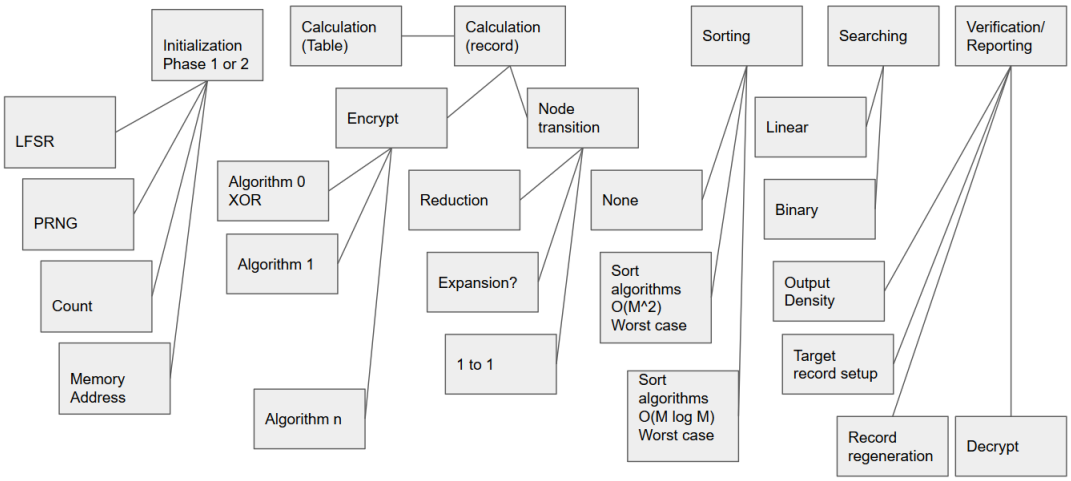

Figure 4.4: Initialization, calculation, sorting, and searching in TMTO-MPI algorithm can be expanded to have more than one implementation

4.5.2 Randomization

Random functions in C

Initial test results using the random C function were not very successful. Even with different seeds there was significant overlap between the generated strings in different cores. Monitoring the randomness as the tables get bigger is problematic to say the least. We decided to go with LFSR as it will hardly overlap and is uniformly distributed.

Uniformly distributed LFSR

In this work we use linear feedback shift register (LFSR) to populate the starting points which generates a set of M records that are uniformly distributed. LFSR sequence will have a period of $2^n - 1$ [67]. The problem with this approach is that it limits the ability of

the program to run in parallel. Haromoto *et al.*[68] proposed an algorithm that would help in making the starting point generation based on LFSR more parallel. Using a fast jump-ahead algorithm they can achieve better results than sliding windows method. LFSR code is based on the C language code in [69],[70]. Katz *et al.*illustrates that a cipher is perfectly indistinguishable from a uniform random distribution of the attacker fails to identify it by a probability highest that 50%. Thus the need of randomization is not important since encryption algorithms output should not be distinguishable from uniformly distributed values.

LFSR caching

Another way to increase the speed of LFSR generation is to generate the values once for the maximum number of cores anticipated once. This can be saved to file and loaded later instead of regeneration. This was not implemented but it could be as part of general pre-emption strategy that would allow the program to stop and pick up where it left.

Using the memory location itself as SP

Instead of generating random numbers for the starting point the program can start with a sequence of numbers which has one to one mapping with the memory locations they are stored in. This can increase the storage capacity to be almost double. The drawback is that it can not be sorted after generating the tables as it will lose the relation to the memory address. This can only be used for attacks with tables that are freshly generated every time.

The need for randomization and or uniform distribution

The researcher believes that setting the starting points using random techniques to be uniformly distributed -as proposed by Hellman's [6] original trade-off - might not be needed as the encryption algorithms should provide this feature organically. All that is needed is for the starting points to be unique to minimize collisions, and that could be easily achieved by a counter or partially copying memory addresses. This should have a significant effect on the performance since the attack will be completely parallel and there will be no chocking point at the beginning. Sykes *et al.*[24] claimed that the attack can be completely parallel which can be consistent with this idea.

4.5.3 Distributing the load

Quota calculation

While Taber [25] focuses on heterogeneous environment where nodes have different memory processor and co processor speeds, that is not the case in this research. Our works is based on GPC cluster that use homogeneous environment. The quota is the same for all cores except the parent core that is dedicated to the first phase of initialization and distributing the load. We decided to go with Coarse-grained parallelism since the solution fits nicely in this manner. To detect false alarms and different leads while trying the initial development of the program we forced the search loops to go all the way to the end, hence all cores should finish in almost the same time.

Send and receive

MPI library takes care of the complex underlying communication system between the nodes. $MPI\_Send()$ sends messages to specific jobs running on other cores identified as ranks. $MPI\_Recv()$ receives those messages. In this work, after initialization first phase the Parent process sends the LFSR results to all the Child processes utilizing a loop that addresses the specific process ID ($rank$). After each child process is done with second phase of initialization, calculation, sorting (if required),

MPI's vector variable

Using MPI standard library we created a different data structure to make sure it is flexible enough to transmit whatever block size required since the framework should work for different types of cryptographic algorithms with different block sizes. The following code define an MPI new data type using MPI_Type_vector to create a vector of $BlockSizeBytes$ bytes ($MPI\_UNSIGNED\_CHAR$).

```
MPI_Type_vector (BlockSizeBytes, 1, 1, MPI_UNSIGNED_CHAR, &BlockSize32);
```

The initial design attempted to create the tables using the Parent process and pass those tables to child processes but this eventually lead to a limitation in memory as it was shared between all processes. Since in this parallel model there is no need for each process to access another process's table we decided to separate the tables completely and only pass the first starting point of the table.

4.5.4 End point calculation

The child processes then start working by completing the initialization. This happens by taking the string that was sent by the parent as the first staring point in the child process table, and using it to generate all the starting points of the table via the same LFSR. At the end of this phase each child process will have an array of strings that has all the starting points. The child process will then proceed to generate the end point for each of the starting point by doing the cryptographic operation and then the reduction function. In this work, even though the reduction function was implemented for earlier implementations for DES it was not used in this scenario since the key size is typically larger than the ciphertext size. On the contrary, the cryptographic operations was changed to take the ciphertext and double it to confirm with the key size.

4.5.5 Sorting

The sorting function was implemented but since the computational complexity of sorting is high $O(n^2)$ and it has no real value in the iterative development approach that was followed. We decided to submit most runs without sorting enabled and measure the effect of sorting on some specific runs. It is worth mentioning that using string comparison - and all string operations- in cryptographic scenarios needs some extra consideration as in C/C++ a string ends with a Null which is not the case in our cryptographic blocks. Two issues might arise: going out of block size boundary while trying to find the null at the end of the string; or matching an early null that was produced by a cryptographic function.

4.5.6 Search

After all the tables are built each child process will search for the expected Ciphertext challenge in the end points. If not found the cryptographic function will be applied to it and search again. This will go for T times (from $t = T$ till $t = 0$) to find all matches and leads. Once a lead is found, the chain is regenerated to the point just before the match was found $t - 1$ to recreate the key. We can then compare the keys and make sure it is the same.

4.5.7 Reporting

All child processes report back to the parent process. All times and cryptographic function outputs are kept in an output file generate by GPC. Each job generates three files: output file, error file, and joblog file. Those files have the results of all processes that were involved in the job. The output file has the output messages created by the program which uses the code in [71] to display the binary representation of number if needed.

4.6 Comparisons with Hellman's original approach

In the following, we highlight the differences between our TMTO-PMI algorithm and the original work by Hellman [6]:

- We used LFSR generated sequences to populate the starting points instead of PRNG to decrease the possibility of overlap.
- No specific reduction function as the key size is larger than the block size.
- While the original Hellman work was linear, this work is almost completely parallel.
- Our work is software based instead of hardware based.
- It is easier to adapted the software to different algorithms since the framework is basically the same and we just need to add the algorithm and match it with a reduction function if needed.
- Our framework does not need permanent storage. This potentially increases the time (LFSR computation as opposed to file(s) read). However, it makes it very flexible in terms of changing plaintext PT, key challenge KC, and ciphertext CT.
- The program does not exit right away on a verified match. It waits for other possible solutions from other cores.

4.7 Design rationale for fine-grained parallel algorithm

To decrease collision, instead of calculating chains while iterating variable t until a distinguished point is reached, or having different reduction functions, one can prepare a doubly

linked list with nodes each representing the start and end point of a range. Each node will have a $from$, to fields representing the beginning and end of range that was already calculated. It will also have two pointers fields essential to a doubly linked list structure. There can be more than one linked list each managed by a different process (running on a specific core) we call *custodian*. Every time a new ciphertext is generated, it is cross checked with the linked list. Figure 4.5 shows the proposed process setup for managing the doubly linked lists. The following results are expected when a child process reports a new value to the linked list custodian:

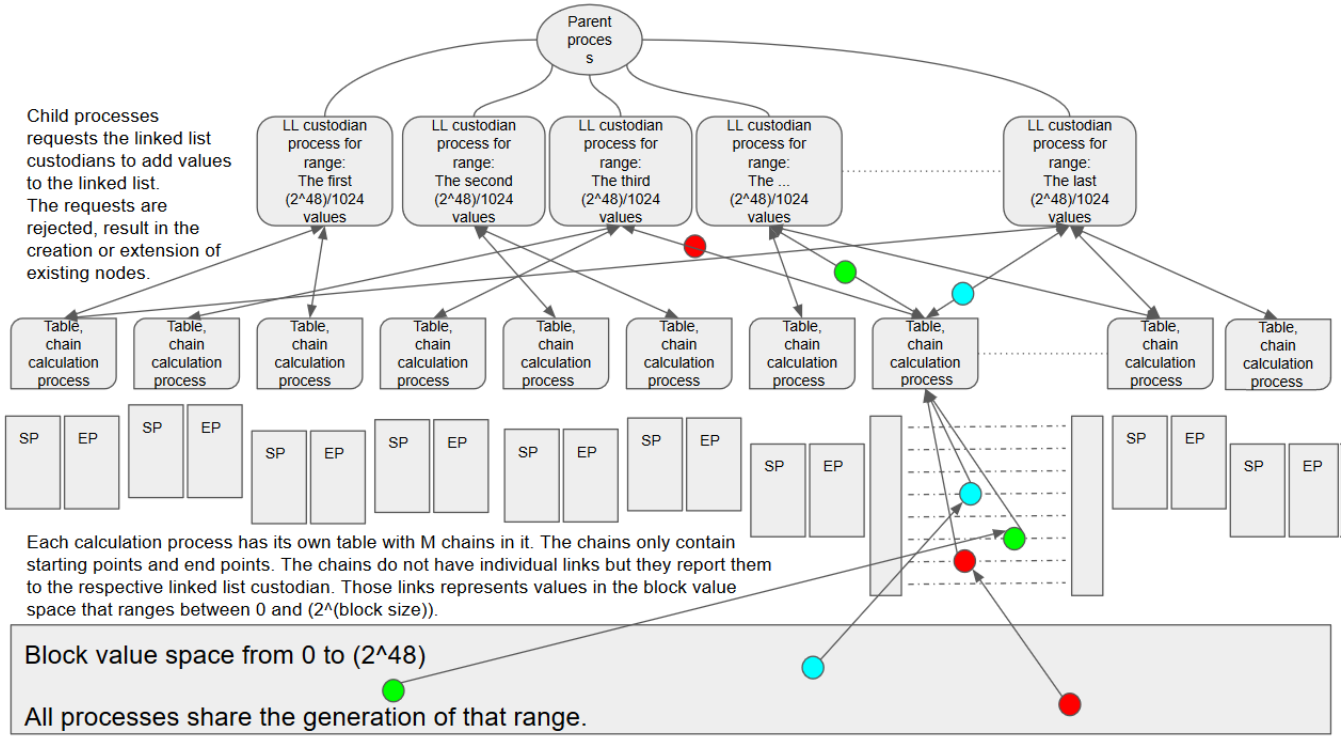

Figure 4.5: Proposed process setup for managing the doubly linked lists in TMTO-MPI algorithm

- If the new ciphertext belongs to one of the existing ranges, then it is not needed as we've reached a chain overlap. The program should discard the result, set $t = t - 1$ and end the chain at the previous cipher test as any further encryption is redundant.
- If the new ciphertext is adjacent to an existing range in the linked list, that linked

list is updated by decreasing the $from$ value by one or increasing the to value by one.

- After a node expands its range in one direction, it is checked with the node just before it and if the touch - become adjacent- they are merged by changing the upper limit of the lower node and discarding the higher node, or by changing the lower limit of the higher node and discarding the lower node. Linked list pointers must be updated accordingly.

- If the new ciphertext is neither part of or adjacent to an existing range, new node is added to the linked list with both the $from$ and to values set to the new ciphertext.

- This technique assumes that all reduction function are the same and will take place after reduction. In this case the linked list must only span the reduced range. Alternatively, if this technique is implemented before the reduction function is applied or if no reduction is applied then the linked list will span the full range of values that can bre presented by the block size.

- The results will differ for different encryption, hashing and reduction algorithms but it is reasonable to expect that the number of nodes in the linked list will decrease or plateau after some time as permutation covered between m and t will start to come closer. This is due to the two mechanisms working to pull in different directions. One mechanism is increasing the number of nodes and another is combining adjacent nodes without increasing the size and possibly decreasing it.

- Starting with starting points' array (SP) that are a sequence of numbers will greatly increase the performance of this technique as we will start with one node in the linked list. If the results are distinguishable from uniformly distributed values it could be an indication of bias in the cipher.

- Another added value for this technique is to measure request traffic on all custodian nodes. If some nodes experience more traffic than others it could indicate a bias in the cipher. Each custodian should keep 4 counter:

  - How many requests were rejected?
  - How many requests caused the creation of a new node?
  - How many requests caused 2 nodes to merge?
  - How many requests caused 1 node to expand its range?

- Having this information for each node and comparing totals and rates for different custodian can reveal a lot about the behavior of a cipher.
- Moving to this fine-grained parallel algorithm as opposed to the current coarse-grained parallel algorithm will increase the overhead and change the complexity. However, it will decrease the collision and add valuable information.
- To make the proposal more efficient we discard the randomization of the starting points.

Figure 4.6 shows the proposed collision detection mechanism using linked list in TMTO-MPI algorithm.

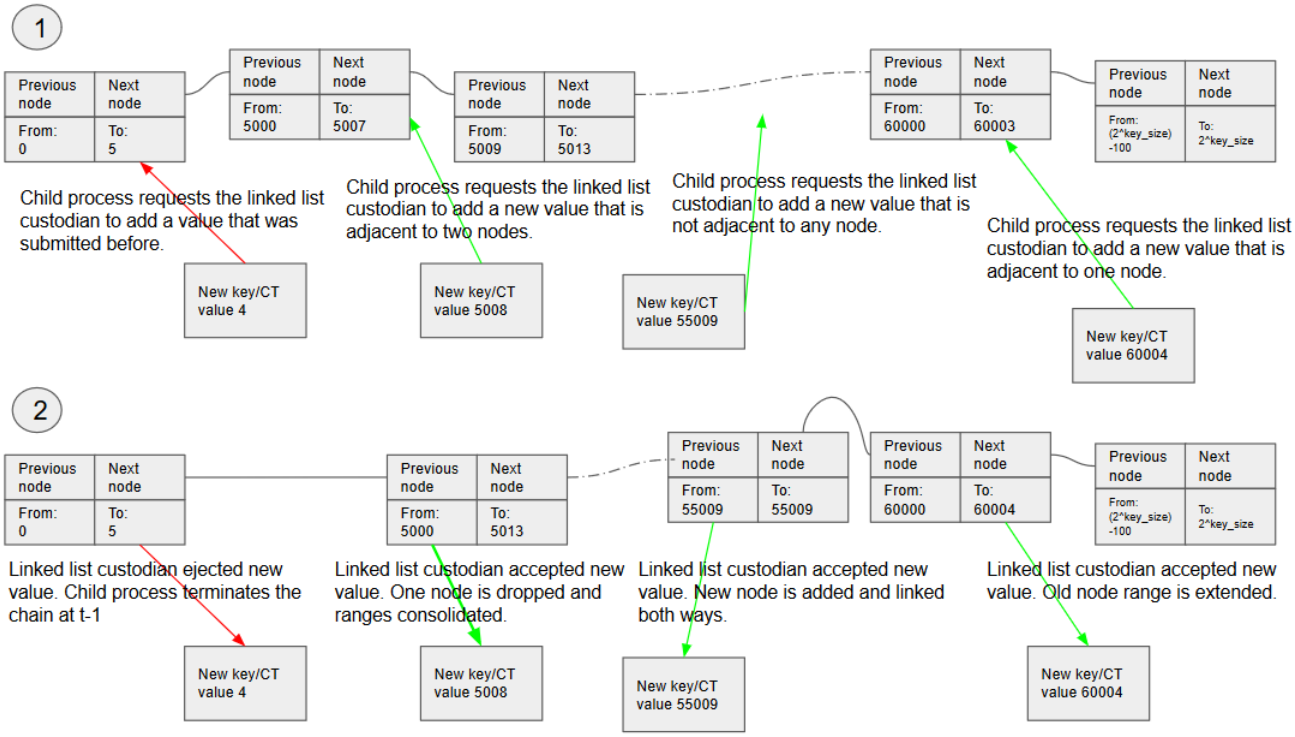

Figure 4.6: Proposed technique for collision detection using MPI base parallel computing and linked list structures

Chapter 5

Analysis of Runs and Results

After receiving the results for the job runs -that will be referred to as jobs throwout this chapter- we tried different approaches to analyze the data. First approach was to extract as much information as possible about every run and present that in spread sheet and try to find relations. The second approach was to tackle individual arbitrary runs and do thorough analysis to find out what happened and note observations. The third approach evolved while working on the second and it focused on clustering similar runs in terms of M, T, and C but with different W and keys. The choice of variables made sense as the results should be very similar with one exception; it could be interrupted if the walltime was not enough. We call this approach (walltime range) clustering approach. The following sections will start with a description of the job evaluation card, then the individual run analysis. And as we realize the walltime range cluster that it belongs to, the cluster will be included to proof the similarity of the results. Finally it will draw some relations from the overall statistics. The analysis script extracted over 1300 job results from the data saved by these jobs. After some manual filtration we had 1270 jobs, 384 out of which completely finished and found the verified the key.

5.1 Sample job evaluation card

Every job or cluster of jobs are described by an evaluation card. The card will contain information about timing, file names, key, plaintext, block size, cipher type, walltime, etc.. The card is usually followed by observations about that job run and its results. The full analysis including the evaluation cards in kept in appendix A. An example of such evalu-

ation card is shown below.

```
Job ID 45697538
The number of booked cores is 1024 (1 Parent and 1023 children)
The job was finished in 0:35:29 minutes but did not find the Key.
Plaintext is: PTPTPT
Random key: [0_53_S][1_39_9][2_46_F][3_4e_N][4_71_q][5_72_r] and it is
   used as the start point of the first chain.
After encrypting the plaintext T times (1024) the chain end point becomes
   : ___[0c]___[12]___[9b]___[41]___[f0]___[6f]
CipherText Challenge: [0_cc_][1_83_~C][2_e1_][3_14_^T][4_4f_O][5_8d_~M]
CT decrypts back to: [0_50_P][1_54_T][2_50_P][3_54_T][4_50_P][5_54_T]
output log: tmto1024t1024c2145386496m2hrs_bestCvrg0.o45697538
Output size (printf overhead): 43101 lines, 3321620 characters.
error log: tmto1024t10.....tCvrg0.e45697538 and it was empty.
Job log: tmt_Fri_Oct_20_..........7538.gpc-sched-ib0.joblog
source code file: tmt_Fri_Oct_20_.........tCvrg0.c
job submission script: tmt_Fri_Oct_20_.......vrg0.sh
object file: tmt_Fri_Oct_20_.............vrg0.out
Cores: 1024
usableCores: 1023
PID:589
Date: tmt_Fri_Oct_20_21_18_33_EDT_2017
M=2145386496   (Note: Total number of records is 2145386496)
C=1024    (Note: 1024 core were used)
W=2:00:00    (Note: Two hours of walltime were booked)
T=1024    (Note: T is 1024)
B=48    (Note: Block size is 48)
E=2    (Note: Cipher is Simeck)
H=0    (Note: Key size and test block size are the same)
L=0    (Note: Initialization was done using LFSR)
S=1    (Note: No sorting)
R=GPC    (Note: This means it was running GPC nodes, not on Development
   node and not in debug mode)
notes=tmto1024t1024c2145386496m2hrs_bestCvrg0.out (Note: This is an
   automated comment generated by a script)
Observations: There was and issue with ........
```

5.2 Individual job and walltime range cluster analysis

In this section we look at some jobs and record observations. We take a look at some successful runs as well as failed runs. We approach individual jobs as well as clusters of similar jobs. We show that some jobs fail due to limited resources while other fail due to bugs. We show from the successful results that that coverage rates support that the algorithm is scalable.

5.2.1 Comparing search coverage rates - 3 billion records $TotalM$

Job 45697540 observation: Key was retrieved in around 53 minutes. This is the worst scenario key retrieval time for this set of key environment variables. Jobs 45697540, 45697541, 45697543, 45697545, 45697546, 45697547, 45697548, and 45697549 have also exhibited very similar behavior using different keys. Applying the same (walltime range) clustering approach we can assume the same behavior would manifest in the following jobs (45697581-45697590, 45697500-45697509, and 45697621-45697630) at a $totalM$ of 3,218,079,744 and T of 1024, after dividing $TotalM$ over 1023 child cores for 3,145,728 records per core, we get a finish time of about 53 minutes which means that all runs with 1, 2, 3, or 4 requested walltime will succeed. The average number of blocks that this algorithm can cover per second is 1,029,876,100 based on the actual walltime that was used after removing two outliers. $CoveragePerSecondBasedOn$ is denoted $CPSBO$.

Figure 5.1 plots this relation. Total coverage is calculated by:

$$
\begin{aligned}
MaximumCoverage &= T * TotalM \\
CPSBOcput &= MaximumCoverage/cput \\
CPSBOUsedWalltime &= MaximumCoverage/(UsedWalltime * C) \\
ValidJobs &= NumberOfJobs - NumberOfOutliers \\
AverageCPSBOUsedWalltime &= \sum_{job=1}^{ValidJobs} CPSBOUsedWalltime_j/ValidJobs
\end{aligned}
$$

5.2.2 Comparing job coverage rates - 1 billion record $TotalM$

Jobs (45697601-45697610, 45697561-45697570, 45697520-45697529, 45697648-45697657, and 45697479-45697488) observations: 1072693248 $TotalM$ with 1023 cores means 1048576 per

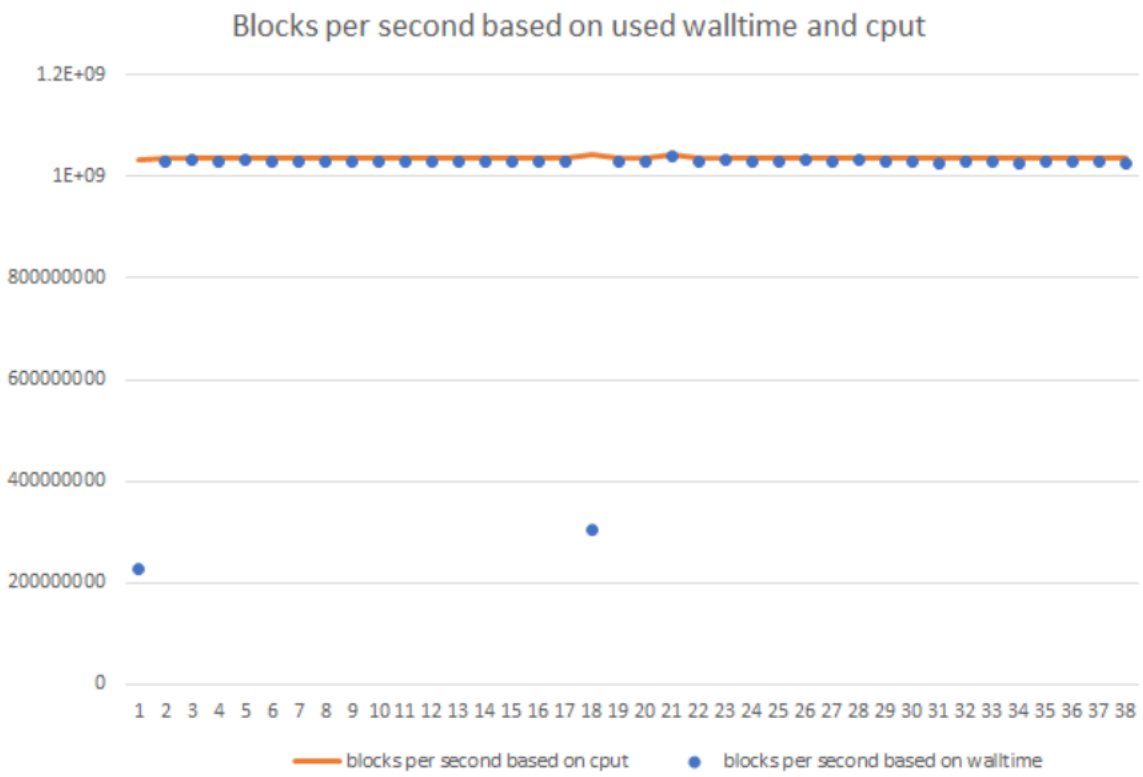

Figure 5.1: The average number of blocks that this algorithm can cover per second is 1,029,876,100

core. All these jobs were successful in a little over 18 minutes. This is consistent with behavior with the "3 million records per core"set. Figure 5.2 plots both sets to illustrate the similar coverage rates. The figures illustrate two methods of calculating coverage per second. The first is by using the used wall time which would be the same value for all cores. The second is by using *cput* which is the sum total of CPU time used by all processors. Except for two outliers the results are consistent. This works in favor of scalability if the framework gets access to more computing resources.

5.2.3 Example of lightweight jobs

Using the same clustering approach we can analyze the following set of jobs together (45697668-45697677, 45697698-45697707, 45697728-45697737, and 45697758-45697767). All

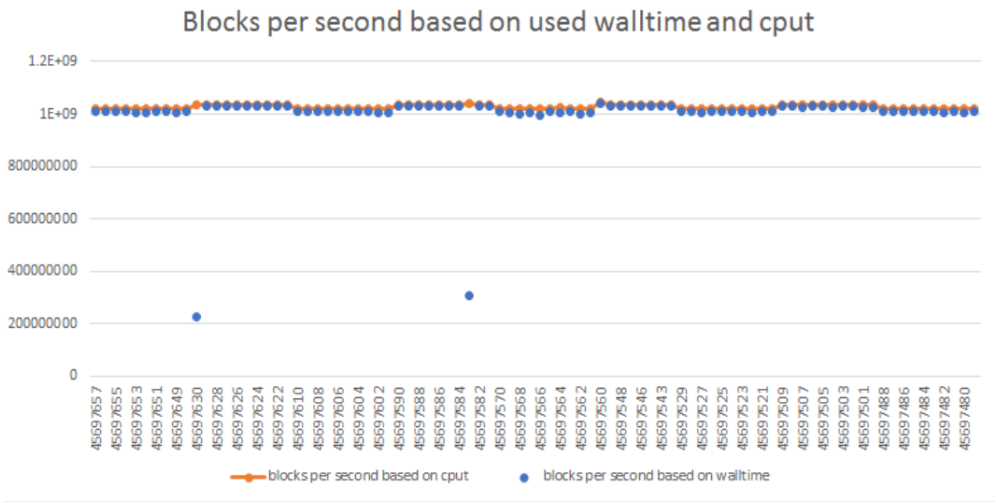

Figure 5.2: Both 1 and 3 million record per core sets behave the in terms of coverage per second

these jobs have the same parameter except for the random key and the requested walltime. Looking at the job runs they all are able to finish and retrieve the random key in about 20 minutes. Booking walltime of 1, 2, 3, and 4 hours is too much for $TotalM$ of 15728640 and T of 1024 since the $TotalM$ after being distributed amongst 1023 cores, is around 15,375 each.

Jobs (45697658-45697667, 45697688-45697697, 45697718-45697727, 45697748-45697757) observation: All those lightweight with about a million record per core and with T at 1024 were successful. 7 cores were usable to the tables out of 8 and $TotalM$ was 7,340,032. Jobs (45697678-45697687, 45697708-45697717, 45697738-45697747, and 45697768-45697777) were also successful using 32 cores (only 31 used for generation and search). $TotalM$ was 32505856 which comes down to 1048576 per core.

5.2.4 Examples of unsuccessful job sets

Job 45697516: The walltime that was booked by the probing script and provided by the GPC was not enough to finish. The job was kicked out while calculating the tables. The overhead of the printing is large and the over all performance can be enhanced by decreasing the output overhead. Other jobs (45697510 to 45697519) behave similarly. This means

it is not a one time problem but rather with the same environment variables it will respond similarly.

Job ID 45697515 is one of them and looking at the details they are more or less the same except the random key is different. The framework failed to find the key even after successfully finishing the search. This could be due to a bug. The generation and negative-result search took about 1 hour 10 minute and was in the scope of the 2 hours walltime booked from the GPC system. Appendix A show the set of jobs that also exhibited and confirmed the same behavior.

Since there seemed to a pattern we tried to group the runs into cluster. This cluster with the faulty results can be extended to cover similar runs with higher walltime. The fact that they fail to reach the results but finish the run gracefully means that nothing is going to change if the run has more walltime. To test this we look at the following jobs (45697631-45697640, and 45697591-45697640). The behavior is consistent except for job 45697640. Considering the whole cluster of jobs, they all finishes in about 1:10 hours if they had more than one hour of walltime booked. If the job had just one hour booked, it is safe to assume that it was killed just around 10 minutes before it was finished. Job 45697640 stands out as an outlier and is an opportunity for more investigation.

Job 45697538 had an issue with the verification code. Jobs (45697530 to 45697537) act similarly where the search is done in 35 minutes but now results were found. No leads or partial matches were found which enforces the conclusion that it was a bug with setting up the random key. Following the (walltime range) clustering method we can assume the same behavior will happen in the following jobs (45697489-45697494, 45697496-45697499, 45697571-45697580, 45697611-45697620, 45786991-45786997, and 45787000). They all finish without being interrupted by GPC in 35 minutes without finding the key.

Jobs (46040577, 46040579, 46040581, and 46040594-46040597) finished without finding the key, exhausting the walltime or issuing an error. Memory limits are assumed to be good since initializing the SP was fine. Looking at the allocated memory, it is about 227BG if we take the median 238314532KB.

5.3 Job coverage in terms of main input variables

5.3.1 Pushing boundaries tactics or probing strategy

Deciding the chain length T and the number of start and end points M to increase the success rate is - according to [25]- based on trial and error. Figure 5.3 shows the general directions of probing that we tried in our work and it focuses on GPC resource boundaries and the planned T and M. Since there are many variables that can be changed, we need to narrow the space and start with some ranges of runs that would give some initial scoping. This initial scoping is only achievable after reaching significant stability in the framework.

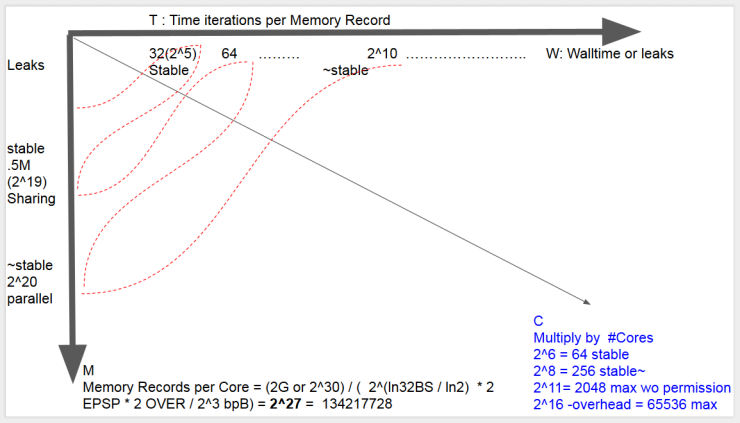

Figure 5.3: Probing the boundaries of GPC

5.3.2 Number of cores coverage

One of the objects of the first approach to analyze the results was to collect some indicators about the coverage of the runs. This helped illustrating how far the capabilities of the system were probed in different directions. Figure 5.4 shows the number of job runs versus the number of cores used. e.g. the second bar means that it used 8 cores to cover about 180 runs. There were only 4 runs that used 64 cores, another 4 runs used 128 cores, and no runs that used 256 cores.

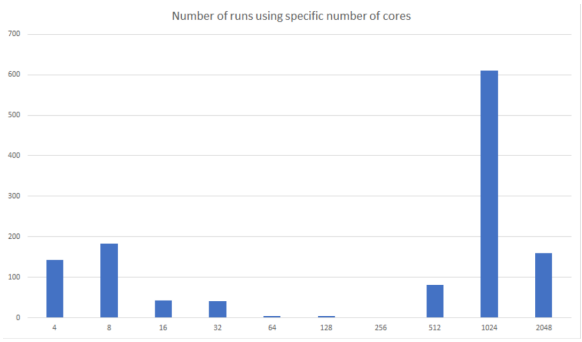

Figure 5.4: Run coverage of number the of cores used

5.3.3 Memory records M coverage

Figure 5.5 shows the number of runs versus the number of memory records $TotalM$ used, while Figure 5.6 shows the number of runs versus the number of memory records M ($TotalM$ per core C) used. Figure 5.7 shows the number of runs versus the number of requested walltime hours, while figure 5.8 shows the number of runs versus the iterations T (in other words rings within chains inside the tables) used.

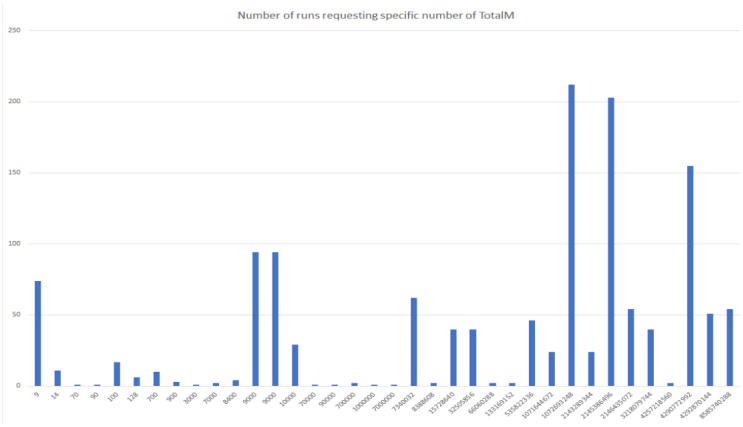

Figure 5.5: Run coverage of the number of $TotalM$ memory records

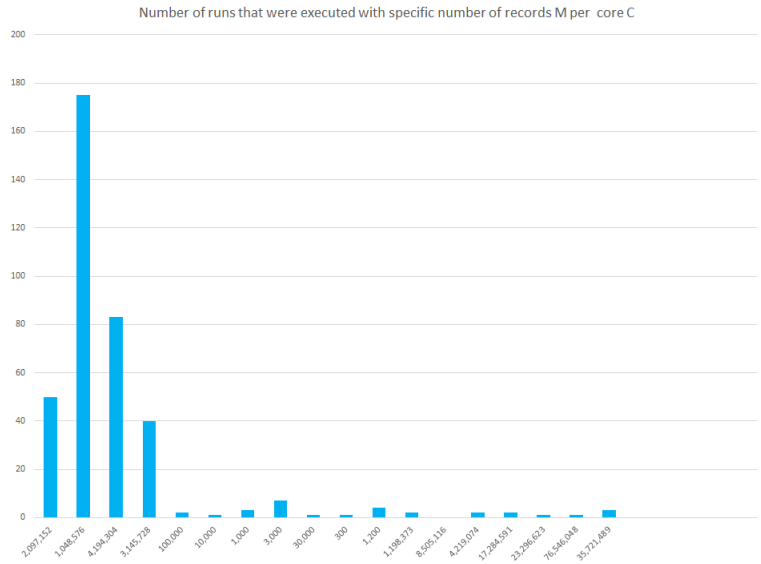

Figure 5.6: Run coverage of the number of M memory records per core

5.4 Measuring changes in program behavior

5.4.1 Decreasing output density

Initial attempt

The output to file, like network and other IO, is handled by MPI. It was important to execute runs that are not output dense and compare the results. We analyze the available data and observe behavior as we exclude some clusters. Figure 5.9 plots all the jobs as follows. Memory records (or number of chains) per core on the left Y axis and denoted MperC is plotted in orange. Output density in bytes divided over CPU time (cput) per core in seconds is plotted in blue on the right Y axis. X axis is the job description. Since the records are sorted from low to high we expect the highest values on the right side. The full data is not useful since it is visually hard to analyze and has invalid records. We exclude the following and plot again in Figure 5.10 :

- Invalid entries (like division by zero due to data extraction problems).
- Records where $TotalM$ is 10,000 records or less.

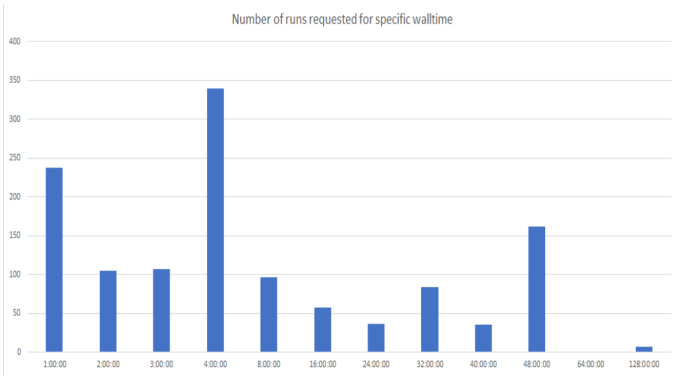

Figure 5.7: Run coverage of the number of requested W walltime hours

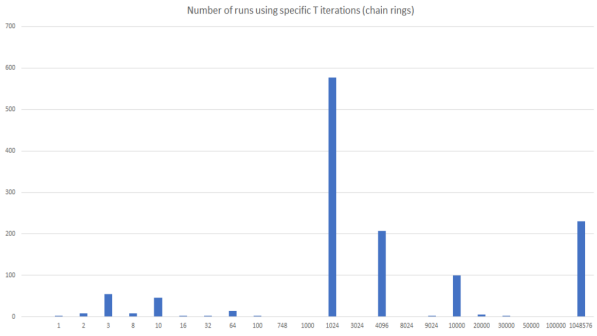

Figure 5.8: Run coverage of the number of requested T iterations

- Records for jobs that aborted and/or failed to recover the validation key.

Figure 5.10 show the same plot after the exclusion. We discover a cluster with jobs that have significantly high output density rate mostly due to high coverage ($T*M$) on the right side. We exclude those and get a more focused data set in Figure 5.11 where we discover two cluster, again on the right side, where we have contradicting behaviors for the relation between MperC and the output density. We observer MperC was just over 3,000,000 while output density was just over 4,000 byte per second per core, and then when MperC was just over 1,000,000 the output density was just under 14,000 byte per second per core. We conclude that the two rates are not corelated and then exclude those clusters. The new subset of data in Figure 5.12 shows that the MperC rate for most jobs is relatively less than that for low output density jobs (which their names start with LowD on the X axis). This becomes more apparent when looking at very low output density jobs (with VLowD at the bedining on the job name on the X axis). Based on this analysis we observe that even though the

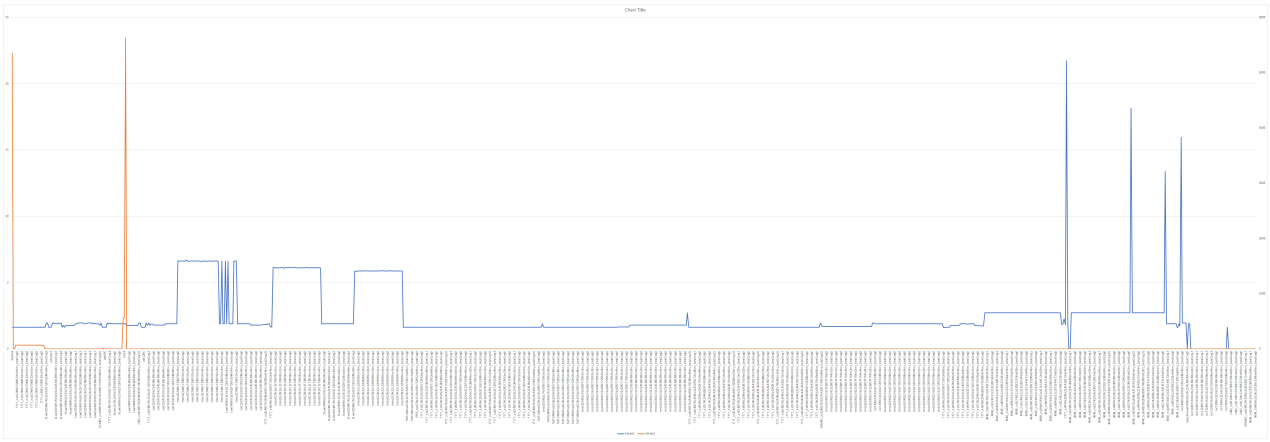

Figure 5.9: Output density in bytes per average cput in seconds versus average M per core for all jobs

MPI I/O operations are transparent to the program they do affect the rate at which we can push it's computation boundary. **Observation:** Some jobs were successful and some were not. Regardless of how much walltime was asked for, the jobs that had T=4096, C=1024, and M=1,072,693,248 finished in around 01:16:20 hours. Some jobs that had T=4096, C=1024, and M=2,145,386,496 stopped after about 04:00:00 hours as the GPC killed the jobs for going over the alloted time. This can be justified as M is twice the size. Looking at the output file (Ex: $LowD4096t1024c2145386496m4hrs_bestCvrg0.o45941114$), the time expires while the child processes are generating the starting point. This sample will be used to decrease the the test density further more and remeasure results.

Some jobs that had T=4096, C=1024, and M=2,145,386,496 Stopped after about 02:31:00 hours. It is not clear why they stopped but they stop around the same place as the other failed jobs. That is, after running LFSR 2,097,152 times for starting points. This prompted the algorithm to decrease output density further more. With 1024 cores we are allowed 16 hours. The whole batch where $TotalM$= 2,145,386,496 records and W = 4:00:00 hours ran out of time. And so did the batch were $TotalM$=1,072,693,248 records and W=4:00:00 hours.

Final attempt

We attempted to radically decrease the debug information especially for the Child processes. Table 5.4.1 illustrates the results. We use 1023 instead of 1024 or 2^{10} since we only

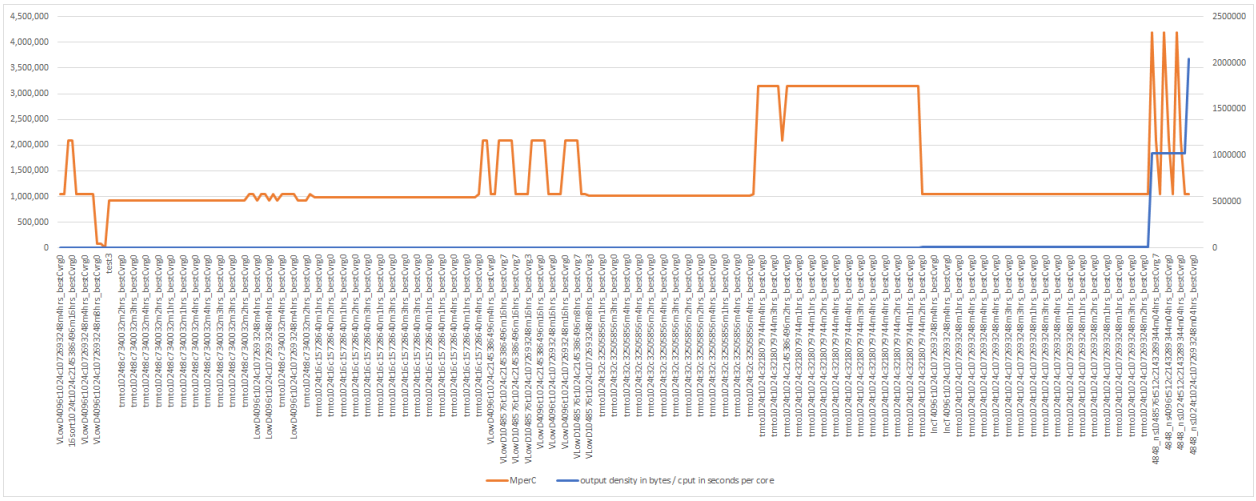

Figure 5.10: Output density in bytes per average cput in seconds versus average M per core for successful jobs

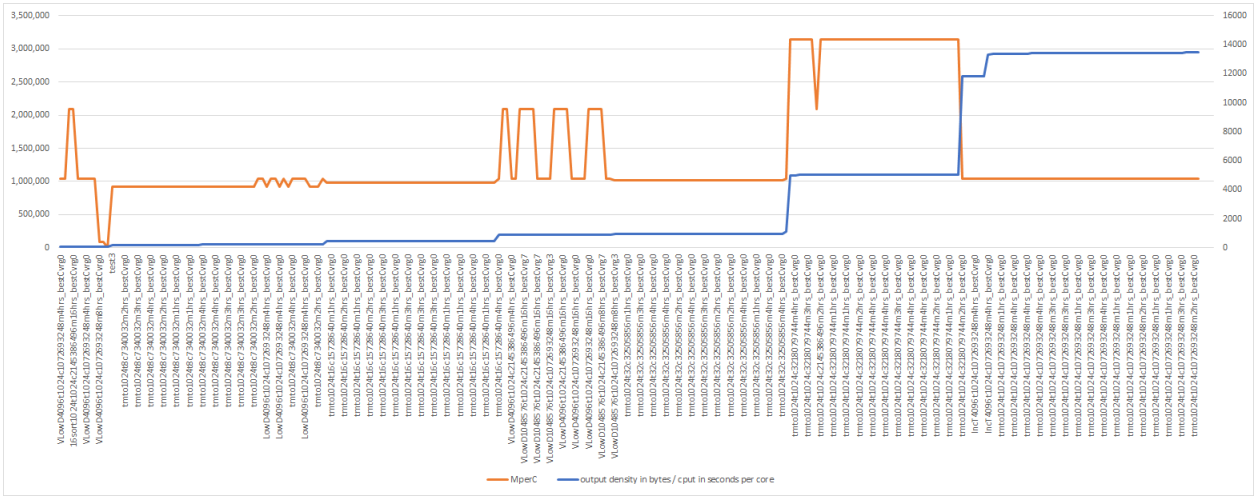

Figure 5.11: Output density in bytes per average cput in seconds versus average M per core for subset 1 of successful jobs

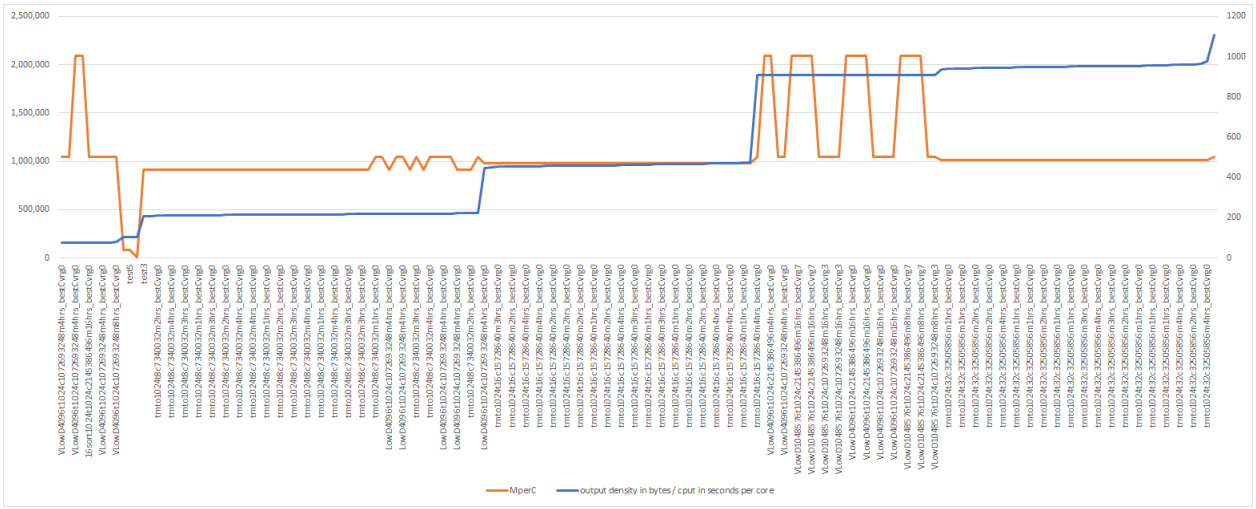

Figure 5.12: Output density in bytes per average cput in seconds versus average M per core for subset 2 of successful jobs

used 1023 child cores for the cryptographic operations and one core is reserver for parent operations.

T	$TotalM$	Observation	Job ID	Search space $\|S\| = TM$
1048576	1,072,693,248	ran out of time (4 hours walltime)	46040582-46040585	$1023 * 2^{40}$
1048576	2,145,386,496	fail the runs	46040586-46040589	$1023 * 2^{41}$
1048576	1,072,693,248	8 hours with smaller M ran out	46040598, 46040599	$1023 * 2^{40}$
4096	1,072,693,248	successful	46040572	$1023 * 2^{32}$

5.4.2 False positives and collisions

Job 46040572 with 4096 T, and 4 hours run was successful. It finished in 1 hour 11 minutes, found the Key but also found 5 false positive leads. Two of the keys were 0x6cc94a0fb8fe, two were 0x3173e31408b3, and one was 0xcb59825b9174. Those false positives were in 5 different chains (different m records) and 2 pairs had the same values. This shows that there is collision between chains. Figure 5.13 reflects the idea that having collision in the false positives space could be an indication that the collision effect is bigger on the tables. Collisions are very costly and there is no way for tracking this while the tables and chains are being generated. We propose the parallel architecture for collision detection and prevention in Section 4.7 for this reason.

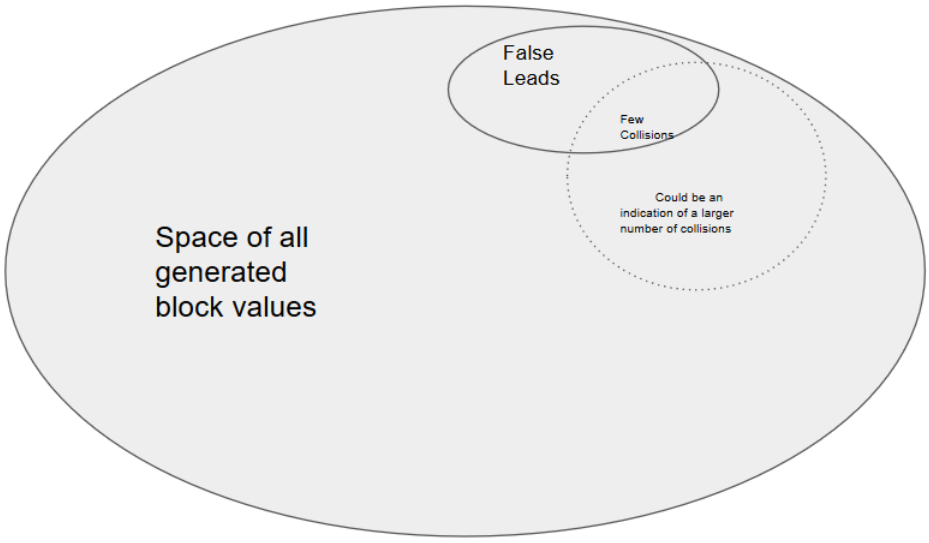

Figure 5.13: Collisions in the false positive space could indicate a much larger collision in the table/chain space

Job 46040573 with the same 4096 T, 4 hours, also finished in about 1 hour 11 minutes, found the Key, and also found 9 false positive leads. Six of the false positives had the value of 0xcc192590b40f which means a high level of collision. There is currently no way to tell exactly what is the percentage of the collision because we do not record it. We only record those that flagged as potential leads and by the definition they are a small percentage. So we can derive that the overall collision is equal to or larger than the lead collision. The

three other false alarms are unique.

Job 46040574 was successful without false positive leads. Job 46040575 found the Key but also found 3 false positive leads. The three false positive leads are unique. Job 46040590 found the Key but also found 9 false positive leads. Two pairs had the same values: 0x031cfabd2efd and 0xb8e6f7103b14, one value was lost in the extraction but is recoverable, and the other 4 had unique values. Job 46040593 found the Key but also found 7 false positive leads, 3 unique, and two pairs are with same value. They all occurred at different t,c, and m.

The most interesting case is job 46040591 which found the key but also found 18 false positive leads. One false positive value (0xc715ed95a111) as a recovered key repeats 6 times in 6 different chains (different m memory records). One repeats 3 times, two repeats twice, and the rest of them are unique. Since this is a significantly larger set of false positives we can compare t, c, and m to see if there is any relations. Figure 5.14 shows that only one value repeats $c = 42$ which means that core 42 yielded 2 false positives at 2 different chains in the same table, chain 1026006 and chain 1639710. Each happened at a different iteration t, respectively, $t = 599$, and $t = 1861$. There are two scales in this figure. The left scale is used to plot c which is the core instance presented by the orange color, and t which is the iteration where the false positive happened and presented by the blue line. The right scale is used to plot m which is the memory record where the false positive happened. For example, the fifth false positive in the diagram show that when t was at 2442 in the 201^{st} core ($c = 201$) a false positive occurred in m (memory record 1825380).
Job 46040592 found the Key but also found 16 false positive leads. One false positive value (0x356d35bca3ef) repeats 12 times in 12 different chains. The remaining 4 false positives had unique values. This is also a significantly large set of false positives. We can compare t, c, and m to see if there is any relations. Figure 5.15 shows that there is no overlap between the false positive coordinates of t, c, and m. Meaning, false positives happened at different iterations t, cores c, memory records m. The same two-scale graph type is used. Left scale is used to plot the core c and the iteration t where the false positive happened at. The right scale is used to plot the memory record m where the false positive happened. For example the 15^{th} false positive happened at $c = 64$, $t = 757$, and $m = 539640$.

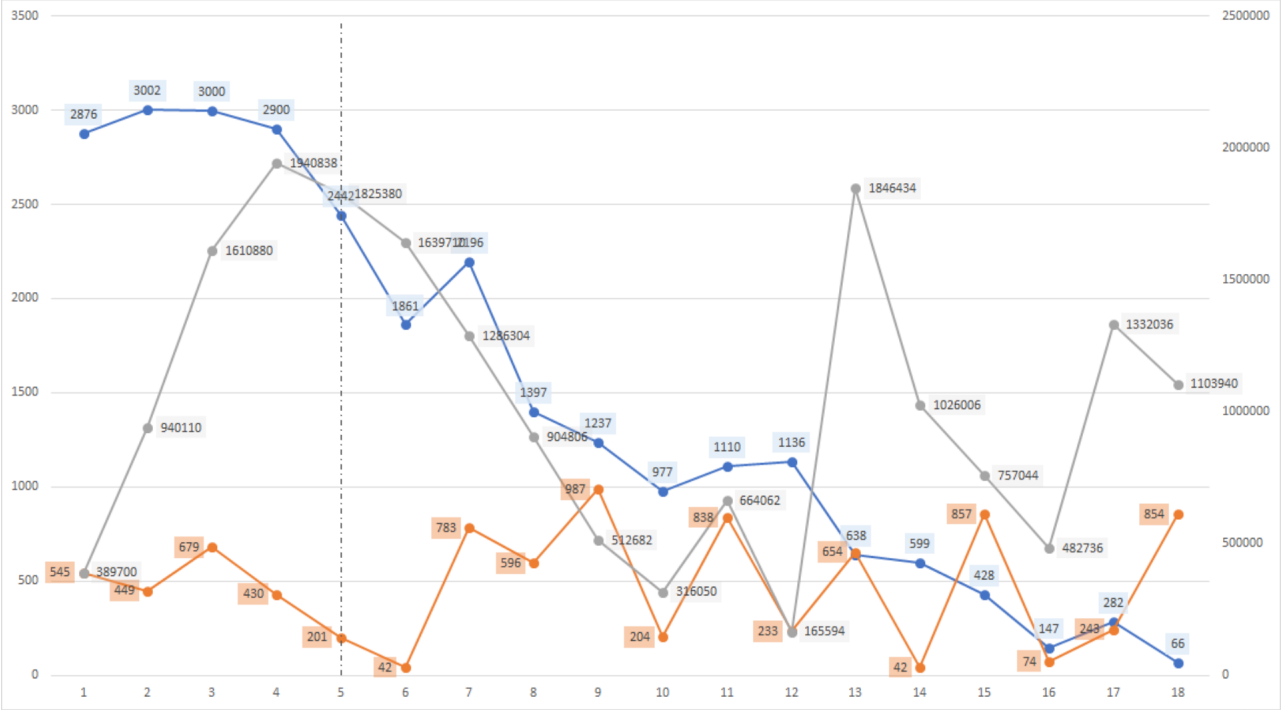

Figure 5.14: False positive plot for job 46040591

5.4.3 Changing encryption algorithms

One of the objectives of the framework was to compare similar runs in terms of key variables using different but comparable encryption algorithms. Since Speck is very similar to Simeck it was an obvious choice. Speck jobs 46336986, 46336988, 46336987, and 46336985 were run using Speck and only one of them succeeded in retrieving the verification key. The rate of false positives was very high compared to Simeck which lead to higher output density and the inability to compare run times.

5.4.4 Measuring the effect of sort

The complexity of sorting algorithms have a significant penalty on the runs that are not particularly useful in this phase of the research as it entails comparing many runs. However, it was also important to sample some runs with sorting enabled to draw some comparisons. The following jobs are comparable to jobs that had high output density for Simeck

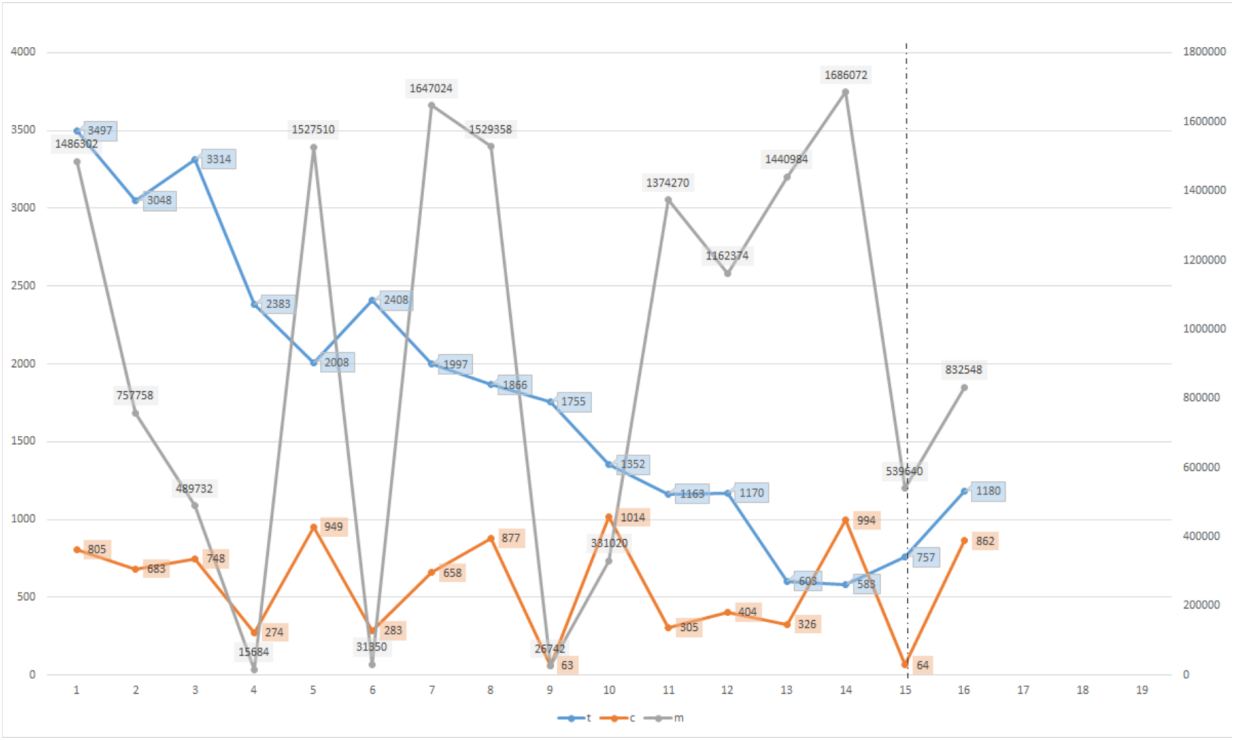

Figure 5.15: False positive plot for job 46040592

(45940946-45940955, 45940966-45940975, 45940956-45940965, 45940976-45940985). The general observation from those jobs that work which used to take a less than two hours, now needs more than 8 hours. Indeed the program gets kicked out by GPC while sorting.

5.5 Search space coverage

Figure 5.16 shows search space coverage per second for all jobs, while Figure 5.17 shows coverage per second for entries that are larger than 7,000,000,000. The plots illustrate that there two major clusters. The relationship between virtual and physical memory is consistent at a ration of about 1.8:1. In general the memory (virtual or real) per coverage decreases as the coverage increases. This could be justified by the limited resources each node has. This simple explanation, however, does not justify the significantly different clusters illustrated for the 1.09844E+12 , 4.39375E+12, and 1.1248E+15 jobs. By coverage

(potential maximum coverage) we mean the maximum number of unique blocks that the job can potentially generate) as a ring within a chain within a table) and then use in the search phase later on in the process.

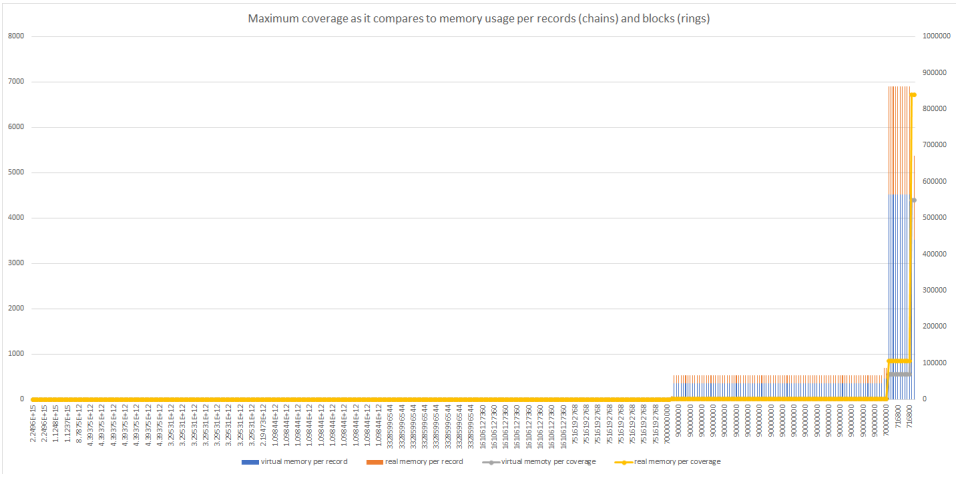

Figure 5.16: Coverage per second for all jobs

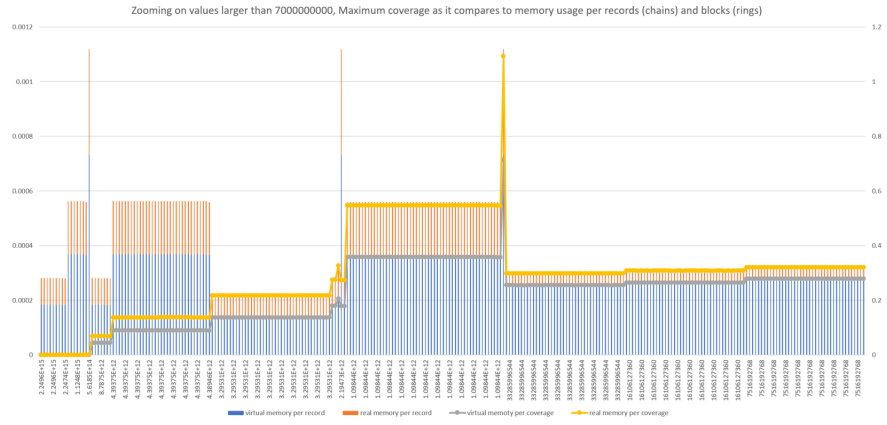

Figure 5.17: Coverage per second for entries that are larger than 7,000,000,00

5.6 Summary of results

To summarize, the TMTO-MPI algorithm exhibits consistency in behavior that means that it would be scalable if more resources are used. The lesser the output density the higher the potential coverage. The output density is a function of debug output and false leads. False positives exist and will differ for different runs. The collision of false positives is an indicator of a potentially larger collision issue. The solution to the collision issue is proposed in Section 4.7. The added complexity of sorting requires higher resources in an actual attack.

Chapter 6

Conclusions and Future Work

6.1 Summary of contribution

6.1.1 Framework development

We have shown in this work that we can build a framework to research TMTO attack against different ciphers in MPI environment called a TMTO-MPI algorithm. Based on the known research in the literature, we define the components of TMTO-MPI algorithm as follows:

1. Individual run creation and submission.
2. Probing different variable via multi-run script.
3. Key variables initialization.
4. Parallel communication.
5. Quota distribution.
6. Starting points random initialization.
7. Calculating endpoints through iterative encryption.
8. Reduction or expansion function.
9. Sorting component.

10. Collision detection or minimization component.
11. Challenge preparation.
12. Saving and loading from disks.
13. Key search.
14. False positive leads handling and key verification.
15. Result analysis.

The algorithm for TMTO-MPI is defined in Section 4.3 which covers the internal and parallel part of the framework. The external part of the framework cover components 1, 2, and 15. Components 10 and 12 were not implemented but we have proposed another algorithm in Section 4.7 to solve this issue. The TMTO problem can be almost completely parallelized. The exception is the first phase of initialization which is used to generate the master LFSR table that is used as a seed for the child processes to generate their own LFSR. Using this technique in this phase is not entirely necessary and we have proposed some other techniques to increase that can increase the performance.

6.1.2 Design Rationale

We identified the differences between this work that is based on parallel computing the original work by Hellman [6] that was not parallel in Section 4.6. We have discussed key concepts, important variables, algorithm, structure, and phases. For example, to build such framework we needed the following two features:

1. **Using XOR for framework robustness.** One of the lessons learned about building a cryptanalysis framework is the use of XOR based mock cipher which can be used to test the framework stability. XOR is very simple and can be adapted to different block sizes. It can provide a good bench mark to compare any cryptographic algorithm against. It will also make the debugging much easier and help detect and verify fixes for memory leaks, out of memory conditions, and segmentation faults.
2. **Verifying algorithm accuracy.** While XOR mock cipher is very good for verifying algorithm stability and scalability it does not help with verifying false positives or that the generation and verification works as expected. The reason is that XOR in this TMTO scenario will keep iterating between two values. We can solve this by injecting the challenge in the tables. We used this extensively for our runs and currently the framework supports injecting the challenge as a starting point.

6.1.3 Testing

We explained our probing technique to test the limitation of this framework and our approach for results analysis (individual, overall and clusters). We've discussed some of the search space coverage rates, successful as well as failed jobs, coverage in terms of main variables, false positives and collisions. The following are some of the relations we've discussed:

1. **Run times for different ciphers.** We expected there would be difference in run times between different ciphers. This indeed happens but mostly -if not solely- due to the effect of different output density caused by different rates of false leads. These results are not conclusive since we only ran few results for Speck but the interesting result here is that: while comparing run times for different ciphers the density of output at the same debug level must be considered. It could eventually prove to be a strength or weakness in the cipher. We modified the input to the two cryptographic algorithms to accept 48 bits as both the key size as well as the plain text and cipher text size.

2. **Limitations.** In this work, the successful tests had a maximum of 2048 cores booked. The GPC and the new Niagara systems allow for more cores to be booked and to increase the coverage of the framework. It is however more important to find ways to measure, understand, and minimize chain collisions before trying to scale the coverage. The reason this has more priority is that by scaling (increasing) the coverage the system might result in more chain collisions which will not even be detectable unless they cause false leads.

3. **Output density.** The density of the output has significant impact on the response time of the run. Since this is a research framework it is expected to have multiple debug levels to stabilize and validate different approaches. However, when the purpose of the run is to calculate and compare timing, it is important to minimize the output or debug level.

6.1.4 Proposal for a fine grained algorithm for collision detection and prevention

While analyzing the run results it became obvious that the most critical point in the success of such attack is collision avoidance. At the beginning of the development phase

we used randomization to initialize the values of the starting points and then we changed the approach to use LFSR to guarantee uniques starting points. We do not have an exact way of measuring collisions but there is an alarming fact that there are many false lead that are equal. Regardless of the root cause of these leads, the fact that they are similar indicate a potential previous and a sure forward chain collision. In the future work we would like to explore the technique proposed in Section 4.7 in Chapter 4, i.e. , proposal to use fine-grained parallel algorithm for effectively detecting and avoiding collisions.

6.2 Future Work

The components defined for the framework are not fully explored. To advance this work we can increase the number of ciphers it supports. We can also write a key-reduction-expansion function. To make the job runs interruptible we can add a feature that supports job preemption by writing to and reading from disk. Support more LFSR polynomials for different sizes of blocks. Support different chain sizes and distinguished points. For the external structure we can add a feature for queue time prediction. Also, adapt to, and use more computing capabilities in the new Niagara system. Most importantly, implementing the new fine grained proposal for TMTO-MPI algorithm.

References

[1] Lidong Chen and Guang Gong. *Communication system security*. CRC press, 2012.

[2] Masumeh Damrudi and Norafida Ithnin. An optimization of tree topology based parallel cryptography. *Mathematical Problems in Engineering*, 2012, 2012.

[3] A. Murat Fiskiran and Ruby B. Lee. Fast parallel table lookups to accelerate symmetric-key cryptography. In *Information Technology: Coding and Computing, 2005. ITCC 2005. International Conference on*, volume 1, pages 526–531. IEEE, 2005.

[4] Sedeeq Hassn Albana Ali Al-Khazraji. Using parallel computing to implement security attack. *International Journal of Computer Science and Information Security*, 13(8):35, 2015.

[5] General purpose cluster (GPC) quickstart. https://wiki.scinet.utoronto.ca/wiki/index.php/GPC_Quickstart.

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

[7] CISSP Steven Hernandez. *Official (ISC) 2 guide to the CISSP CBK*. CRC Press, 2009.

[8] Philippe Oechslin. Making a faster cryptanalytic time-memory trade-off. In *Annual International Cryptology Conference*, pages 617–630. Springer, 2003.

[9] François-Xavier Standaert, Gael Rouvroy, Jean-Jacques Quisquater, and Jean-Didier Legat. A time-memory tradeoff using distinguished points: New analysis & fpga results. In *CHES*, volume 2523, pages 593–609. Springer, 2002.

[10] Mark Stamp. Once upon a time-memory tradeoff. *San Jose State University, Department of Computer Science*, 2003.

[11] Alex Biryukov. Some thoughts on time-memory-data tradeoffs. *IACR Cryptology ePrint Archive*, 2005:207, 2005.

[12] Jin Hong, Kyung Chul Jeong, Eun Young Kwon, In-Sok Lee, and Daegun Ma. Variants of the distinguished point method for cryptanalytic time memory trade-offs. In *International Conference on Information Security Practice and Experience*, pages 131–145. Springer, 2008.

[13] Gildas Avoine, Pascal Junod, and Philippe Oechslin. Time-memory trade-offs: False alarm detection using checkpoints. In *INDOCRYPT*, volume 3797, pages 183–196. Springer, 2005.

[14] Arvind Narayanan and Vitaly Shmatikov. Fast dictionary attacks on passwords using time-space tradeoff. In *Proceedings of the 12th ACM conference on Computer and communications security*, pages 364–372. ACM, 2005.

[15] Alex Biryukov, Sourav Mukhopadhyay, and Palash Sarkar. Improved time-memory trade-offs with multiple data. In *Selected Areas in Cryptography*, volume 3897, pages 110–127. Springer, 2005.

[16] Nele Mentens, Lejla Batina, Bart Preneel, and Ingrid Verbauwhede. Time-memory trade-off attack on FPGA platforms: Unix password cracking. *ARC*, 3985:323–334, 2006.

[17] Elad Barkan, Eli Biham, and Adi Shamir. Rigorous bounds on cryptanalytic time/memory tradeoffs. In *CRYPTO*, volume 4117, pages 1–21. Springer, 2006.

[18] Daegun Ma and Jin Hong. Success probability of the hellman trade-off. *Information Processing Letters*, 109(7):347–351, 2009.

[19] Jin Hong. The cost of false alarms in hellman and rainbow tradeoffs. *Designs, Codes and Cryptography*, 57(3):293–327, 2010.

[20] Ehud D. Karnin. A parallel algorithm for the knapsack problem. *IEEE Transactions on Computers*, (5):404–408, 1984.

[21] Hamid Reza Amirazizi and Martin E. Hellman. Time-memory-processor trade-offs. *IEEE Transactions on Information Theory*, 34(3):505–512, 1988.

[22] Henry Ker-Chang Chang, Jonathan Jen-Rong Chen, and Shyong-Jian Shyu. A parallel algorithm for the knapsack problem using a generation and searching technique. *Parallel Computing*, 20(2):233–243, 1994.

[23] Ken-Li Li, Ren-Fa Li, and Qing-Hua Li. Optimal parallel algorithms for the knapsack problem without memory conflicts. *Journal of Computer Science and Technology*, 19(6):760–768, 2004.

[24] Edward R. Sykes and Wesley Skoczen. An improved parallel implementation of rainbowcrack using mpi. *Journal of Computational Science*, 5(3):536–541, 2014.

[25] Michael S. Taber. Distributed pre-computation for a cryptanalytic time-memory trade-off. 2008.

[26] Amos Fiat and Moni Naor. Rigorous time/space tradeoffs for inverting functions. In *Proceedings of the twenty-third annual ACM symposium on Theory of computing*, pages 534–541. ACM, 1991.

[27] Markku-Juhani Olavi Saarinen. A time-memory tradeoff attack against lili-128. In *International Workshop on Fast Software Encryption*, pages 231–236. Springer, 2002.

[28] T.E. Bjørstad. Cryptanalysis of grain using time/memory/data tradeoffs. *on Estream Phase*, 3, 2013.

[29] Itai Dinur. Cryptanalytic time-memory-data tradeoffs for fx-constructions with applications to prince and pride. In *EUROCRYPT (1)*, pages 231–253, 2015.

[30] Alex Biryukov and Adi Shamir. Cryptanalytic time/memory/data tradeoffs for stream ciphers. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 1–13. Springer, 2000.

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

[32] Vrizlynn L.L. Thing and Hwei-Ming Ying. A novel time-memory trade-off method for password recovery. *digital investigation*, 6:S114–S120, 2009.

[33] Dorothy Elizabeth Robling Denning. *Cryptography and data security*. Addison-Wesley Longman Publishing Co., Inc., 1982.

[34] Johan Borst, Bart Preneel, and Joos Vandewalle. On the time-memory tradeoff between exhaustive key search and table precomputation. In *Symposium on Information Theory in the Benelux*, pages 111–118. Technische Universiteit Delft, 1998.

[35] Jean-Jacques Quisquater, Francois-Xavier Standaert, Gael Rouvroy, Jean-Pierre David, and Jean-Didier Legat. A cryptanalytic time-memory tradeoff: First fpga implementation. In *FPL*, volume 2438, pages 780–789. Springer, 2002.

[36] Gildas Avoine, Pascal Junod, and Philippe Oechslin. Characterization and improvement of time-memory trade-off based on perfect tables. *ACM Transactions on Information and System Security (TISSEC)*, 11(4):17, 2008.

[37] Jin Hong and Palash Sarkar. Rediscovery of time memory tradeoffs. *IACR Cryptology ePrint Archive*, 2005:90, 2005.

[38] Gangqiang Yang, Bo Zhu, Valentin Suder, Mark D. Aagaard, and Guang Gong. The simeck family of lightweight block ciphers. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 307–329. Springer, 2015.

[39] Message passing interface. https://en.wikipedia.org/wiki/Message_Passing_Interface.

[40] Bo Zhu. Reference code for the simeck family of block ciphers. https://github.com/bozhu/Simeck, 2015.

[41] Ray Beaulieu, Stefan Treatman-Clark, Douglas Shors, Bryan Weeks, Jason Smith, and Louis Wingers. The SIMON and SPECK lightweight block ciphers. In *Design Automation Conference (DAC), 2015 52nd ACM/EDAC/IEEE*, pages 1–6. IEEE, 2015.

[42] Sebastian Gesemann. Reference code for the speck family of block ciphers. http://github.com/GaloisInc/saw-script/blob/master/examples/simon-speck, 2013.

[43] Nurdan Saran and Ali Doganaksoy. Choosing parameters to achieve a higher success rate for hellman time memory trade off attack. In *Availability, Reliability and Security, 2009. ARES'09. International Conference on*, pages 504–509. IEEE, 2009.

[44] Ruakh. Replace one substring for another string in shell script. https://stackoverflow.com/questions/13210880/replace-one-substring-for-another-string-in-shell-script, 2012.

[45] Norman Ramsey. How to get wc -l to print just the number of lines without file name? http://stackoverflow.com/questions/10238363/how-to-get-wc-l-to-print-just-the-number-of-lines-without-file-name, 2012.

[46] Rob I. How to split one string into multiple variables in bash shell? https://stackoverflow.com/questions/10520623/how-to-split-one-string-into-multiple-variables-in-bash-shell, 2012.

[47] Johannes Schaub. How do I split a string on a delimiter in bash. http://stackoverflow.com/questions/918886/how-do-i-split-a-string-on-a-delimiter-in-bash, 2009.

[48] tr. https://ss64.com/bash/tr.html.

[49] user3101398. How to decrypt simple XOR encryption. http://stackoverflow.com/questions/20579363/how-to-decrypt-simple-xor-encryption, 2013.

[50] Paul Griffiths. How to convert a char array to a uint16_t by casting type pointer? http://stackoverflow.com/questions/27558956/how-to-convert-a-char-array-to-a-uint16-t-by-casting-type-pointer, 2014.

[51] Steven Sudit. Convert a uint16-t to char[2] to be sent over socket (unix). http://stackoverflow.com/questions/13279024/convert-a-uint16-t-to-char2-to-be-sent-over-socket-unix, 2012.

[52] C passing array of structure to function. http://www.c4learn.com/c-programming/c-passing-array-of-structure-to-function.

[53] strcmp. http://en.cppreference.com/w/c/string/byte/strcmp.

[54] Ofir Carny. How to convert integer to char in C? http://stackoverflow.com/questions/2279379/how-to-convert-integer-to-char-in-c, 2010.

[55] Qwertz. How to convert integer to string in C? http://stackoverflow.com/questions/9655202/how-to-convert-integer-to-string-in-c, 2012.

[56] John Bode. How do i create an array of strings in C? http://stackoverflow.com/questions/1088622/how-do-i-create-an-array-of-strings-in-c, 2009.

[57] Alexandre Jasmin. Dynamic memory for 2D char array. http://stackoverflow.com/questions/2614249/dynamic-memory-for-2d-char-array, 2010.

[58] Francis Lavergne. Dynamic memory allocation in MPI. http://stackoverflow.com/questions/25628321/dynamic-memory-allocation-in-mpi, 2014.

[59] Jonathan Dursi. Sending and receiving 2D array over MPI. http://stackoverflow.com/questions/5901476/sending-and-receiving-2d-array-over-mpi, 2011.

[60] James Hu. How to sort an array of string alphabetically (case sensitive, nonstandard collation). http://stackoverflow.com/questions/12646734/how-to-sort-an-array-of-string-alphabetically-case-sensitive-nonstandard-colla, 2014.

[61] fopen. http://www.cplusplus.com/reference/cstdio/fopen.

[62] strcat. http://www.cplusplus.com/reference/cstring/strcat.

[63] Dwhitney67. Get process ID in C. https://ubuntuforums.org/showthread.php?t=1430052, 2010.

[64] Manish Bhojasia. C program to print the program name and all its arguments. http://www.sanfoundry.com/c-program-print-program-name-and-arguments.

[65] C library function - strcmp(). https://www.tutorialspoint.com/c_standard_library/c_function_strcmp.htm.

[66] Forrest Hoffman. Using derived data types with MPI. http://www.linux-mag.com/id/1332, 2003.

[67] Solomon W Golomb and Guang Gong. *Signal design for good correlation: for wireless communication, cryptography, and radar*. Cambridge University Press, 2005.

[68] Hiroshi Haramoto, Makoto Matsumoto, and Pierre L'Ecuyer. A fast jump ahead algorithm for linear recurrences in a polynomial space. *Lecture Notes in Computer Science*, 5203:290, 2008.

[69] Linear-feedback shift register. https://en.wikipedia.org/wiki/Linear-feedback_shift_register.

[70] Axel Kemper. Galois LFSR explanation of code. http://stackoverflow.com/questions/16891655/galois-lfsr-explanation-of-code, 2013.

[71] Paxdiablo. Display the binary representation of a number in C? http://stackoverflow.com/questions/699968/display-the-binary-representation-of-a-number-in-c, 2009.

[72] Jonathan Katz and Yehuda Lindell. *Introduction to modern cryptography*. CRC press, 2014.

[73] Charles P. Pfleeger and Shari Lawrence Pfleeger. *Security in computing*. Prentice Hall Professional Technical Reference, 2002.

[74] Sabari Pramanik and Sanjit Kumar Setua. Dna cryptography. In *Electrical & Computer Engineering (ICECE), 2012 7th International Conference on*, pages 551–554. IEEE, 2012.

[75] Jin Hong and Palash Sarkar. New applications of time memory data tradeoffs. In *Asiacrypt*, volume 3788, pages 353–372. Springer, 2005.

[76] Sourav Mukhopadhyay and Palash Sarkar. Application of LFSRs in time/memory trade-off cryptanalysis. In *International Workshop on Information Security Applications*, pages 25–37. Springer, 2005.

APPENDICES

Appendix A

Result Analysis

A.1 Arbitrary sample and cluster analysis

A.1.1 The first batch

Job ID 45697540

```
The number of booked cores is 1024 (1 Parent and 1023 children)
The job was finished properly
-- lead at t=1 core=1, m=0, key 1st B_l_
-- Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
-- Found CT:_[ ][98]__[[][5b]__[][1c]__[][06]__[][bb]__[ ][ed]_
-- Recovered Key(EP-1):_[l][6c]__[p][70]__[L][4c]__[f][66]__[N][4e]__
   [8][38]_
-- Chain SP:_[l][6c]__[p][70]__[L][4c]__[f][66]__[N][4e]__[8][38]_
-- Orig CT:_[9][39]__[][1e]__[  ][09]__[  ][91]__[][01]__[ ][e8]_
-- Orig Key:_[l][6c]__[p][70]__[L][4c]__[f][66]__[N][4e]__[8][38]_
-- CT 4 verify:_[9][39]__[][1e]__[ ][09]__[][91]__[][01]__[][e8]_
-- We have Verification also.

plaintext is: PTPTPT
Random key: [0_6c_l][1_70_p][2_4c_L][3_66_f][4_4e_N][5_38_8] and it is
   used as the start point of the first chain.
```

```
After encrypting the plaintext T times (1024) the chain end point becomes
   : ___[98]___[5b]___[1c]___[06]___[bb]___[ed]
Ciphertext Challenge: [0_39_9][1_1e_][2_09_ ][3 _91_ ][4_01_][5 _e8_ ]
CT decrypts back to: [0_50_P][1_54_T][2_50_P][3_54_T][4_50_P][5_54_T]
output log: tmto1024t1024c3218079744m2hrs_bestCvrg0.o45697540
Output size (printf overhead): 315279 lines , 15924859 characters.
error log: tmto1024t1024c3218079744m2hrs_bestCvrg0.e45697540
with some deprecation warnings "Using mlock ulimits for SHM_HUGETLB is
   deprecated"
Job log: tmt_Fri_Oct_20_21_18_35_EDT_2017_M=3218079744_C=1024_W=2_00_00_T
   =1024_B=48_E=2_H=0_L=0_S=1_
R=GPC__notes=tmto1024t1024c3218079744m2hrs_bestCvrg0_45697540.gpc-sched-
   ib0.joblog
source code file: tmt_Fri_Oct_20_21_18_35_EDT_2017_M=3218079744_C=1024_W=2
   _00_00_T=1024_B=48_E=2_H=0_L=0_S=1_
R=GPC__notes=tmto1024t1024c3218079744m2hrs_bestCvrg0.c
job submission script: tmt_Fri_Oct_20_21_18_35_EDT_2017_M=3218079744_C
   =1024_W=2_00_00_T=1024_B=48_E=2_H=0_L=0_S=1_
R=GPC__notes=tmto1024t1024c3218079744m2hrs_bestCvrg0.out
object file: tmt_Fri_Oct_20_21_18_35_EDT_2017_M=3218079744_C=1024_W=2
   _00_00_T=1024_B=48_E=2_H=0_L=0_S=1_
R=GPC__notes=tmto1024t1024c3218079744m2hrs_bestCvrg0.sh
Cores: 1024
usableCores:1023
PID:31650
Creation Date: Fri_Oct_20_21_18_35_EDT_2017
Run time: 00:53:16
M=3218079744
C=1024
W=2:00:00
T=1024
B=48
E=2
H=0
L=0
S=1
R=GPC
notes=tmto1024t1024c3218079744m2hrs_bestCvrg0
```

Observation: Key was retrieved in around 53 minutes. This is the worst scenario key retrieval time for this set of key environment variables. Jobs 45697540, 45697541, 45697543, 45697545, 45697546, 45697547, 45697548, and 45697549 have also exhibited very similar behavior using different keys.

```
o45697540:Limits:     neednodes=128:ppn=8,nodes=128:ppn=8,walltime=02:00:00
o45697540:Resources:  cput=903:39:05,mem=266017656kb,vmem=452404292kb,
   walltime=00:53:16
o45697541:Limits:     neednodes=128:ppn=8,nodes=128:ppn=8,walltime=02:00:00
o45697541:Resources:  cput=903:37:22,mem=266008612kb,vmem=452404244kb,
   walltime=00:53:19
o45697542:Limits:     neednodes=128:ppn=8,nodes=128:ppn=8,walltime=02:00:00
o45697542:Resources:  cput=903:39:57,mem=265915088kb,vmem=452404140kb,
   walltime=00:53:19
o45697543:Limits:     neednodes=128:ppn=8,nodes=128:ppn=8,walltime=02:00:00
o45697543:Resources:  cput=903:36:16,mem=265928668kb,vmem=452404252kb,
   walltime=00:53:17
o45697544:Limits:     neednodes=128:ppn=8,nodes=128:ppn=8,walltime=02:00:00
o45697544:Resources:  cput=903:41:04,mem=266047884kb,vmem=452404296kb,
   walltime=00:53:23
o45697545:Limits:     neednodes=128:ppn=8,nodes=128:ppn=8,walltime=02:00:00
o45697545:Resources:  cput=903:33:23,mem=265994232kb,vmem=452404280kb,
   walltime=00:53:15
o45697546:Limits:     neednodes=128:ppn=8,nodes=128:ppn=8,walltime=02:00:00
o45697546:Resources:  cput=903:42:33,mem=266056828kb,vmem=452404268kb,
   walltime=00:53:18
o45697547:Limits:     neednodes=128:ppn=8,nodes=128:ppn=8,walltime=02:00:00
o45697547:Resources:  cput=903:45:46,mem=265968096kb,vmem=452404184kb,
   walltime=00:53:22
o45697548:Limits:     neednodes=128:ppn=8,nodes=128:ppn=8,walltime=02:00:00
o45697548:Resources:  cput=903:37:26,mem=266036156kb,vmem=452404232kb,
   walltime=00:53:15
o45697549:Limits:     neednodes=128:ppn=8,nodes=128:ppn=8,walltime=02:00:00
o45697549:Resources:  cput=903:35:55,mem=266041348kb,vmem=452404052kb,
   walltime=00:53:19

tmto1024t1024c3218079744m2hrs_bestCvrg0.o45697540:
```

```
lead at t=1 core=1, m=0, key1stB_l_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[ ][98]__[[][5b]__[][1c]__[][06]__[][bb]__[ ][ed]_
Recovered Key(EP-1):_[l][6c]__[p][70]__[L][4c]__[f][66]__[N][4e]__[8][38]_
Chain SP:_[l][6c]__[p][70]__[L][4c]__[f][66]__[N][4e]__[8][38]_
Orig CT:_[9][39]__[][1e]__[ ][09]__[ ][91]__[][01]__[ ][e8]_
Orig Key:_[l][6c]__[p][70]__[L][4c]__[f][66]__[N][4e]__[8][38]_
CT 4 verify:_[9][39]__[][1e]__[ ][09]__[ ][91]__[][01]__[ ][e8]_
We have Verification also.

tmto1024t1024c3218079744m2hrs_bestCvrg0.o45697541:
lead at t=1 core=1, m=0, key1stB_P_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[ ][80]__[][19]__[ ][b5]__[G][47]__[ ][c2]__[ ][af]_
Recovered Key(EP-1):_[P][50]__[J][4a]__[S][53]__[c][63]__[i][69]__[N][4e]_
Chain SP:_[P][50]__[J][4a]__[S][53]__[c][63]__[i][69]__[N][4e]_
Orig CT:_[F][46]__[ ][e4]__[][02]__[][0e]__[][f6]__[ ][b2]_
Orig Key:_[P][50]__[J][4a]__[S][53]__[c][63]__[i][69]__[N][4e]_
CT 4 verify:_[F][46]__[ ][e4]__[][02]__[][0e]__[][f6]__[ ][b2]_
We have Verification also.

tmto1024t1024c3218079744m2hrs_bestCvrg0.o45697542:
lead at t=1 core=1, m=0, key1stB_Q_ Orig PT:_[P][50]__[T][54]__[P][50]__[T
   ][54]__[P][50]__[T][54]_
Found CT:_[ ][a7]__[ ][fd]__[ ][86]__[ ][8b]__[ ][b1]__[ ][ec]_
Recovered Key(EP-1):_[Q][51]__[0][30]__[H][48]__[6][36]__[z][7a]__[9][39]_
Chain SP:_[Q][51]__[0][30]__[H][48]__[6][36]__[z][7a]__[9][39]_
Orig CT:_[ ][ac]__[ ][8c]__[!][21]__[)][29]__[q][71]__[v][76]_
Orig Key:_[Q][51]__[0][30]__[H][48]__[6][36]__[z][7a]__[9][39]_
CT 4 verify:_[ ][ac]__[ ][8c]__[!][21]__[)][29]__[q][71]__[v][76]_
We have Verification also.

tmto1024t1024c3218079744m2hrs_bestCvrg0.o45697543:Yaaaay lead at t=1 core
   =1, m=0, key1stB_a_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[ ][eb]__[ ][be]__[][1a]__[ ][f0]__[z][7a]__[ ][09]_
Recovered Key(EP-1):_[a][61]__[R][52]__[r][72]__[n][6e]__[K][4b]__[p][70]_
Chain SP:_[a][61]__[R][52]__[r][72]__[n][6e]__[K][4b]__[p][70]_
```

```
Orig CT:_[  ][a7]__[  ][ee]__[d][64]__[ ][c5]__[  ][a4]__[  ][86]_
Orig Key:_[a][61]__[R][52]__[r][72]__[n][6e]__[K][4b]__[p][70]_
CT 4 verify:_[ ][a7]__[  ][ee]__[d][64]__[ ][c5]__[  ][a4]__[  ][86]_
We have Verification also.

tmto1024t1024c3218079744m2hrs_bestCvrg0.o45697544:
lead at t=1 core=1, m=0, key1stB_m_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[  ][ef]__[][1c]__[-][2d]__[][b5]__[  ][b9]__[ ][20]_
Recovered Key(EP-1):_[m][6d]__[m][6d]__[Q][51]__[k][6b]__[d][64]__[x][78]_
Chain SP:_[m][6d]__[m][6d]__[Q][51]__[k][6b]__[d][64]__[x][78]_
Orig CT:_[  ][8d]__[w][77]__[ ][e8]__[  ][9f]__[][07]__[{][7b]_
Orig Key:_[m][6d]__[m][6d]__[Q][51]__[k][6b]__[d][64]__[x][78]_
CT 4 verify:_[ ][8d]__[w][77]__[ ][e8]__[  ][9f]__[][07]__[{][7b]_
We have Verification also.

tmto1024t1024c3218079744m2hrs_bestCvrg0.o45697545:
lead at t=1 core=1, m=0, key1stB_7_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[  ][ff]__[N][4e]__[ ][86]__[  ][ae]__[  ][80]__[  ][80]_
Recovered Key(EP-1):_[7][37]__[P][50]__[3][33]__[Y][59]__[6][36]__[j][6a]_
Chain SP:_[7][37]__[P][50]__[3][33]__[Y][59]__[6][36]__[j][6a]_
Orig CT:_[H][48]__[I][49]__[][a3]__[  ][95]__[][03]__[+][2b]_
Orig Key:_[7][37]__[P][50]__[3][33]__[Y][59]__[6][36]__[j][6a]_
CT 4 verify:_[H][48]__[I][49]__[][a3]__[ ][95]__[][03]__[+][2b]_
We have Verification also.

tmto1024t1024c3218079744m2hrs_bestCvrg0.o45697546:][0a]_
Recovered Key(EP-1):_[S][53]__[j][6a]__[Q][51]__[8][38]__[W][57]__[6][36]_
Chain SP:_[S][53]__[j][6a]__[Q][51]__[8][38]__[W][57]__[6][36]_
Orig CT:_[:][3a]__[O][4f]__[][06]__[0][30]__[,][2c]__[][b7]_
Orig Key:_[S][53]__[j][6a]__[Q][51]__[8][38]__[W][57]__[6][36]_
CT 4 verify:_[:][3a]__[O][4f]__[][06]__[0][30]__[,][2c]__[][b7]_
We have Verification also.

tmto1024t1024c3218079744m2hrs_bestCvrg0.o45697547:
lead at t=1 core=1, m=0, key1stB_v_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
```

```
Found CT:_[Q][51]__[ ][99]__[  ][b5]__[i][69]__[O][4f]__[][89]_
Recovered Key(EP-1):_[v][76]__[x][78]__[1][31]__[c][63]__[w][77]__[s][73]_
Chain SP:_[v][76]__[x][78]__[1][31]__[c][63]__[w][77]__[s][73]_
Orig CT:_[][0b]__[][04]__[][f6]__[][14]__[ ][c5]__[s][73]_
Orig Key:_[v][76]__[x][78]__[1][31]__[c][63]__[w][77]__[s][73]_
CT 4 verify:_[][0b]__[][04]__[][f6]__[][14]__[ ][c5]__[s][73]_
We have Verification also.

tmto1024t1024c3218079744m2hrs_bestCvrg0.o45697548:
lead at t=1 core=1, m=0, key1stB_b_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[  ][c5]__[u][75]__[ ][d1]__[  ][a7]__[  ][c9]__[  ][fa]_
Recovered Key(EP-1):_[b][62]__[g][67]__[v][76]__[x][78]__[w][77]__[x][78]_
Chain SP:_[b][62]__[g][67]__[v][76]__[x][78]__[w][77]__[x][78]_
Orig CT:_[;][3b]__[!][21]__[-][2d]__[u][75]__[][c7]__[][ef]_
Orig Key:_[b][62]__[g][67]__[v][76]__[x][78]__[w][77]__[x][78]_
CT 4 verify:_[;][3b]__[!][21]__[-][2d]__[u][75]__[][c7]__[][ef]_
We have Verification also.

tmto1024t1024c3218079744m2hrs_bestCvrg0.o45697549:
lead at t=1 core=1, m=0, key1stB_c_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[)][29]__[ ][88]__[3][33]__[&][26]__[][dc]__[  ][ee]_
Recovered Key(EP-1):_[c][63]__[s][73]__[m][6d]__[8][38]__[6][36]__[X][58]_
Chain SP:_[c][63]__[s][73]__[m][6d]__[8][38]__[6][36]__[X][58]_
Orig CT:_[ ][20]__[y][79]__[][92]__[  ][a3]__[  ][d1]__[  ][8c]_
Orig Key:_[c][63]__[s][73]__[m][6d]__[8][38]__[6][36]__[X][58]_
CT 4 verify:_[ ][20]__[y][79]__[][92]__[ ][a3]__[  ][d1]__[  ][8c]_
We have Verification also.
```

Applying the same (walltime range) clustering approach we can assume the same behavior would manifest in the following jobs (o45697630 45697629 45697628 45697627 45697626 45697624 45697625 45697623 45697621 45697622 45697590 45697589 45697588 45697583 45697587 45697586 45697585 45697584 45697582 45697581 45697509 45697508 45697507 45697505 45697506 45697504 45697503 45697502 45697501 45697500).

```
45697630
=>> PBS: job killed: walltime 14428 exceeded limit 14400
```

```
Lead at t=1 core=1, m=0, key1stB_B_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[M-\][dc]__[M-‘][e0]__[M-A][c1]__[M-A][c1]__[M-K][cb]__[M-’][a7]
   _
Recovered Key(EP-1):_[B][42]__[B][42]__[X][58]__[C][43]__[k][6b]__[g][67]_
Chain SP:_[B][42]__[B][42]__[X][58]__[C][43]__[k][6b]__[g][67]_
Orig CT:_[7][37]__[a][61]__[M-‘][e0]__["][22]__[8][38]__[/][2f]_
Orig Key:_[B][42]__[B][42]__[X][58]__[C][43]__[k][6b]__[g][67]_
CT 4 verify:_[7][37]__[a][61]__[M-‘][e0]__["][22]__[8][38]__[/][2f]_
We have Verification also.
Limits:               nednodes=128:ppn=8,nodes=128:ppn=8,walltime=04:00:00
Resources:     cput=907:20:05,mem=266211824kb,vmem=452404332kb,walltime
   =04:00:28

45697629
Lead at t=1 core=1, m=0, key1stB_j_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[a][61]__[M-$][a4]__[M-}][fd]__[M-F][c6]__[X][58]__[M-j][ea]_
Recovered Key(EP-1):_[j][6a]__[Z][5a]__[Z][5a]__[q][71]__[1][31]__[V][56]_
Chain SP:_[j][6a]__[Z][5a]__[Z][5a]__[q][71]__[1][31]__[V][56]_
Orig CT:_[M-^P][90]__[M-^?][ff]__[_][5f]__[M-^F][86]__[6][36]__[M-^^][9e]_
Orig Key:_[j][6a]__[Z][5a]__[Z][5a]__[q][71]__[1][31]__[V][56]_
CT 4 verify:_[M-^P][90]__[M-^?][ff]__[_][5f]__[M-^F][86]__[6][36]__[M
   -^^][9e]_
We have Verification also.
Limits:               nednodes=128:ppn=8,nodes=128:ppn=8,walltime=04:00:00
Resources:     cput=903:48:44,mem=265919208kb,vmem=452404204kb,walltime
   =00:53:18

45697628
Lead at t=1 core=1, m=0, key1stB_k_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[M-s][f3]__[M-i][e9]__[<][3c]__[M-G][c7]__[>][3e]__[\][5c]_
Recovered Key(EP-1):_[k][6b]__[b][62]__[d][64]__[7][37]__[t][74]__[5][35]_
Chain SP:_[k][6b]__[b][62]__[d][64]__[7][37]__[t][74]__[5][35]_
Orig CT:_[^E][05]__[\][5c]__[I][49]__[M-Y][d9]__[M-n][ee]__[<][3c]_
Orig Key:_[k][6b]__[b][62]__[d][64]__[7][37]__[t][74]__[5][35]_
CT 4 verify:_[^E][05]__[\][5c]__[I][49]__[M-Y][d9]__[M-n][ee]__[<][3c]_
```

```
We have Verification also.
Limits:               neednodes=128:ppn=8,nodes=128:ppn=8,walltime=04:00:00
Resources:     cput=903:46:58,mem=265849824kb,vmem=452404264kb,walltime
   =00:53:16

45697627
Lead at t=1 core=1, m=0, key1stB_z_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[M-^F][86]__[+][2b]__[^V][16]__[M-I][c9]__[W][57]__[!][21]_
Recovered Key(EP-1):_[z][7a]__[3][33]__[i][69]__[D][44]__[5][35]__[z][7a]_
Chain SP:_[z][7a]__[3][33]__[i][69]__[D][44]__[5][35]__[z][7a]_
Orig CT:_[M-^D][84]__[^O][0f]__[M-^O][8f]__[M-^W][97]__[M-^^][9e]__[M-L][
   cc]_
Orig Key:_[z][7a]__[3][33]__[i][69]__[D][44]__[5][35]__[z][7a]_
CT 4 verify:_[M-^D][84]__[^O][0f]__[M-^O][8f]__[M-^W][97]__[M-^^][9e]__[M-
   L][cc]_
We have Verification also.
Limits:               neednodes=128:ppn=8,nodes=128:ppn=8,walltime=04:00:00
Resources:     cput=903:50:35,mem=266069728kb,vmem=452404268kb,walltime
   =00:53:23

45697626
Lead at t=1 core=1, m=0, key1stB_S_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[M-Q][d1]__[M-L][cc]__[m][6d]__[M-"][a2]__[^\][1c]__[|][7c]_
Recovered Key(EP-1):_[S][53]__[Y][59]__[j][6a]__[9][39]__[8][38]__[Q][51]_
Chain SP:_[S][53]__[Y][59]__[j][6a]__[9][39]__[8][38]__[Q][51]_
Orig CT:_[T][54]__[M-^N][8e]__[-][2d]__[M-b][e2]__[^O][0f]__[M-X][d8]_
Orig Key:_[S][53]__[Y][59]__[j][6a]__[9][39]__[8][38]__[Q][51]_
CT 4 verify:_[T][54]__[M-^N][8e]__[-][2d]__[M-b][e2]__[^O][0f]__[M-X][d8]_
We have Verification also.
Limits:               neednodes=128:ppn=8,nodes=128:ppn=8,walltime=04:00:00
Resources:     cput=903:45:58,mem=265853272kb,vmem=452404244kb,walltime
   =00:53:14

45697624
Lead at t=1 core=1, m=0, key1stB_e_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
```

```
Found CT:_[M-4][b4]__[D][44]__[M-"][a2]__[^H][08]__[Y][59]__[M-Z][da]_
Recovered Key(EP-1):_[e][65]__[S][53]__[f][66]__[J][4a]__[j][6a]__[G][47]_
Chain SP:_[e][65]__[S][53]__[f][66]__[J][4a]__[j][6a]__[G][47]_
Orig CT:_[u][75]__[^W][17]__[A][41]__[M-G][c7]__[^[][1b]__[M-j][ea]_
Orig Key:_[e][65]__[S][53]__[f][66]__[J][4a]__[j][6a]__[G][47]_
CT 4 verify:_[u][75]__[^W][17]__[A][41]__[M-G][c7]__[^[][1b]__[M-j][ea]_
We have Verification also.
Limits:               neednodes=128:ppn=8,nodes=128:ppn=8,walltime=04:00:00
Resources:     cput=903:53:08,mem=265893924kb,vmem=452404232kb,walltime
   =00:53:18

45697625
Lead at t=1 core=1, m=0, key1stB_N_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[^E][05]__[M-+][ab]__[^_][1f]__[~][7e]__[^X][18]__[M-a][e1]_
Recovered Key(EP-1):_[N][4e]__[N][4e]__[i][69]__[R][52]__[Z][5a]__[f][66]_
Chain SP:_[N][4e]__[N][4e]__[i][69]__[R][52]__[Z][5a]__[f][66]_
Orig CT:_[^F][06]__[M-g][e7]__[M-,][ac]__[]][5d]__[^T][14]__[M-^Q][91]_
Orig Key:_[N][4e]__[N][4e]__[i][69]__[R][52]__[Z][5a]__[f][66]_
CT 4 verify:_[^F][06]__[M-g][e7]__[M-,][ac]__[]][5d]__[^T][14]__[M-^Q][91]
   _
We have Verification also.
Limits:               neednodes=128:ppn=8,nodes=128:ppn=8,walltime=04:00:00
Resources:     cput=903:50:07,mem=266072216kb,vmem=452404272kb,walltime
   =00:53:17

45697623
Lead at t=1 core=1, m=0, key1stB_G_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[)][29]__[M-3][b3]__[J][4a]__[^Q][11]__[M-^?][ff]__[M-^F][86]_
Recovered Key(EP-1):_[G][47]__[h][68]__[D][44]__[H][48]__[H][48]__[I][49]_
Chain SP:_[G][47]__[h][68]__[D][44]__[H][48]__[H][48]__[I][49]_
Orig CT:_[Z][5a]__[M-K][cb]__[J][4a]__[M-T][d4]__[3][33]__[i][69]_
Orig Key:_[G][47]__[h][68]__[D][44]__[H][48]__[H][48]__[I][49]_
CT 4 verify:_[Z][5a]__[M-K][cb]__[J][4a]__[M-T][d4]__[3][33]__[i][69]_
We have Verification also.
Limits:               neednodes=128:ppn=8,nodes=128:ppn=8,walltime=04:00:00
Resources:     cput=903:56:23,mem=265907196kb,vmem=452404276kb,walltime
```

```
   =00:53:19

45697621
Lead at t=1 core=1, m=0, key1stB_K_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[^W][17]__["][22]__[M-^Q][91]__[U][55]__[~][7e]__[Z][5a]_
Recovered Key(EP-1):_[K][4b]__[s][73]__[u][75]__[x][78]__[v][76]__[7][37]_
Chain SP:_[K][4b]__[s][73]__[u][75]__[x][78]__[v][76]__[7][37]_
Orig CT:_[M-R][d2]__[M-v][f6]__[-][2d]__[Z][5a]__[*][2a]__[8][38]_
Orig Key:_[K][4b]__[s][73]__[u][75]__[x][78]__[v][76]__[7][37]_
CT 4 verify:_[M-R][d2]__[M-v][f6]__[-][2d]__[Z][5a]__[*][2a]__[8][38]_
We have Verification also.
Limits:               nednodes=128:ppn=8,nodes=128:ppn=8,walltime=04:00:00
Resources:     cput=904:31:42,mem=265852316kb,vmem=452404332kb,walltime
   =00:53:23

45697622
Lead at t=1 core=1, m=0, key1stB_s_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[M-^J][8a]__[y][79]__[%][25]__[M-T][d4]__[M-[][db]__[M-U][d5]_
Recovered Key(EP-1):_[s][73]__[e][65]__[B][42]__[6][36]__[i][69]__[A][41]_
Chain SP:_[s][73]__[e][65]__[B][42]__[6][36]__[i][69]__[A][41]_
Orig CT:_[M-O][cf]__[D][44]__[M-=][bd]__[L][4c]__[>][3e]__[M-O][cf]_
Orig Key:_[s][73]__[e][65]__[B][42]__[6][36]__[i][69]__[A][41]_
CT 4 verify:_[M-O][cf]__[D][44]__[M-=][bd]__[L][4c]__[>][3e]__[M-O][cf]_
We have Verification also.
Limits:               nednodes=128:ppn=8,nodes=128:ppn=8,walltime=04:00:00
Resources:     cput=903:50:09,mem=265841520kb,vmem=452404240kb,walltime
   =00:53:17

45697590
Lead at t=1 core=1, m=0, key1stB_x_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[J][4a]__[M-C][c3]__[M-T][d4]__[ ][20]__[M-|][fc]__[M-^G][87]_
Recovered Key(EP-1):_[x][78]__[a][61]__[7][37]__[d][64]__[G][47]__[7][37]_
Chain SP:_[x][78]__[a][61]__[7][37]__[d][64]__[G][47]__[7][37]_
Orig CT:_[j][6a]__[M-^E][85]__[M-^K][8b]__[M-^Y][99]__[M-^@][80]__[M-M][cd
   ]_
```

```
Orig Key:_[x][78]__[a][61]__[7][37]__[d][64]__[G][47]__[7][37]_
CT 4 verify:_[j][6a]__[M-^E][85]__[M-^K][8b]__[M-^Y][99]__[M-^@][80]__[M-M
   ][cd]_
We have Verification also.
Limits:               neednodes=128:ppn=8,nodes=128:ppn=8,walltime=03:00:00
Resources:     cput=903:58:11,mem=265841784kb,vmem=452404048kb,walltime
   =00:53:18

45697589
Lead at t=1 core=1, m=0, key1stB_B_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[M-s][f3]__[M-e][e5]__[M-Z][da]__[M-o][ef]__[M-W][d7]__[M-^F
   ][86]_
Recovered Key(EP-1):_[B][42]__[D][44]__[y][79]__[E][45]__[N][4e]__[L][4c]_
Chain SP:_[B][42]__[D][44]__[y][79]__[E][45]__[N][4e]__[L][4c]_
Orig CT:_[M-_][df]__[M-;][bb]__[7][37]__[i][69]__[M-u][f5]__[M-n][ee]_
Orig Key:_[B][42]__[D][44]__[y][79]__[E][45]__[N][4e]__[L][4c]_
CT 4 verify:_[M-_][df]__[M-;][bb]__[7][37]__[i][69]__[M-u][f5]__[M-n][ee]_
We have Verification also.
Limits:               neednodes=128:ppn=8,nodes=128:ppn=8,walltime=03:00:00
Resources:     cput=903:55:42,mem=266022484kb,vmem=452404132kb,walltime
   =00:53:22

45697588
Lead at t=1 core=1, m=0, key1stB_h_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[^[][1b]__[M-E][c5]__[F][46]__[L][4c]__[^A][01]__[;][3b]_
Recovered Key(EP-1):_[h][68]__[S][53]__[S][53]__[h][68]__[C][43]__[U][55]_
Chain SP:_[h][68]__[S][53]__[S][53]__[h][68]__[C][43]__[U][55]_
Orig CT:_[^L][0c]__[M-(][a8]__[M->][be]__[^?][7f]__[M-^][de]__[E][45]_
Orig Key:_[h][68]__[S][53]__[S][53]__[h][68]__[C][43]__[U][55]_
CT 4 verify:_[^L][0c]__[M-(][a8]__[M->][be]__[^?][7f]__[M-^][de]__[E][45]_
We have Verification also.
Limits:               neednodes=128:ppn=8,nodes=128:ppn=8,walltime=03:00:00
Resources:     cput=903:51:14,mem=265858864kb,vmem=452404272kb,walltime
   =00:53:20

45697583
```

```
=>> PBS: job killed: walltime 10806 exceeded limit 10800
Lead at t=1 core=1, m=0, key1stB_3_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[M-C][c3]__[M-^G][87]__[M-}][fd]__[M-^K][8b]__[[][5b]__[M-^?][ff
   ]_
Recovered Key(EP-1):_[3][33]__[k][6b]__[2][32]__[u][75]__[3][33]__[1][31]_
Chain SP:_[3][33]__[k][6b]__[2][32]__[u][75]__[3][33]__[1][31]_
Orig CT:_[t][74]__[M-^V][96]__[y][79]__[M-O][cf]__[^R][12]__[M-^][de]_
Orig Key:_[3][33]__[k][6b]__[2][32]__[u][75]__[3][33]__[1][31]_
CT 4 verify:_[t][74]__[M-^V][96]__[y][79]__[M-O][cf]__[^R][12]__[M-^][de]_
We have Verification also.
Limits:               nednodes=128:ppn=8,nodes=128:ppn=8,walltime=03:00:00
Resources:     cput=899:28:04,mem=266064568kb,vmem=452404268kb,walltime
   =03:00:08

45697587
Lead at t=1 core=1, m=0, key1stB_c_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[M-F][c6]__[B][42]__[M-=][bd]__[L][4c]__[M-s][f3]__[M-"][a2]_
Recovered Key(EP-1):_[c][63]__[z][7a]__[A][41]__[Q][51]__[n][6e]__[l][6c]_
Chain SP:_[c][63]__[z][7a]__[A][41]__[Q][51]__[n][6e]__[l][6c]_
Orig CT:_[M-^?][ff]__[^N][0e]__[P][50]__[M-1][b1]__[M-l][ec]__[M-#][a3]_
Orig Key:_[c][63]__[z][7a]__[A][41]__[Q][51]__[n][6e]__[l][6c]_
CT 4 verify:_[M-^?][ff]__[^N][0e]__[P][50]__[M-1][b1]__[M-l][ec]__[M-#][a3
   ]_
We have Verification also.
Limits:               nednodes=128:ppn=8,nodes=128:ppn=8,walltime=03:00:00
Resources:     cput=903:51:46,mem=265907808kb,vmem=452404208kb,walltime
   =00:53:21

45697586
Lead at t=1 core=1, m=0, key1stB_e_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[M-^T][94]__[M-D][c4]__[M-G][c7]__[^Z][1a]__[M-^Y][99]__[/][2f]_
Recovered Key(EP-1):_[e][65]__[N][4e]__[7][37]__[k][6b]__[I][49]__[Q][51]_
Chain SP:_[e][65]__[N][4e]__[7][37]__[k][6b]__[I][49]__[Q][51]_
Orig CT:_[^F][06]__[M-P][d0]__[p][70]__[#][23]__[M-T][d4]__[M-^Y][99]_
Orig Key:_[e][65]__[N][4e]__[7][37]__[k][6b]__[I][49]__[Q][51]_
```

```
CT 4 verify:_[^F][06]__[M-P][d0]__[p][70]__[#][23]__[M-T][d4]__[M-^Y][99]_
We have Verification also.
Limits:               nednodes=128:ppn=8,nodes=128:ppn=8,walltime=03:00:00
Resources:     cput=903:52:46,mem=265946256kb,vmem=452404244kb,walltime
   =00:53:19

45697585
Lead at t=1 core=1, m=0, key1stB_U_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[M-S][d3]__[M-O][cf]__[M-Y][d9]__[n][6e]__[3][33]__[M-P][d0]_
Recovered Key(EP-1):_[U][55]__[A][41]__[9][39]__[M][4d]__[a][61]__[q][71]_
Chain SP:_[U][55]__[A][41]__[9][39]__[M][4d]__[a][61]__[q][71]_
Orig CT:_[M-^R][92]__[M-^V][96]__[>][3e]__[7][37]__[M-'][a7]__[h][68]_
Orig Key:_[U][55]__[A][41]__[9][39]__[M][4d]__[a][61]__[q][71]_
CT 4 verify:_[M-^R][92]__[M-^V][96]__[>][3e]__[7][37]__[M-'][a7]__[h][68]_
We have Verification also.
Limits:               nednodes=128:ppn=8,nodes=128:ppn=8,walltime=03:00:00
Resources:     cput=903:57:36,mem=265979608kb,vmem=452404232kb,walltime
   =00:53:20

45697584
Lead at t=1 core=1, m=0, key1stB_u_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[M-Z][da]__[M-v][f6]__[^V][16]__[d][64]__[M-*][aa]__[M-6][b6]_
Recovered Key(EP-1):_[u][75]__[m][6d]__[C][43]__[f][66]__[I][49]__[O][4f]_
Chain SP:_[u][75]__[m][6d]__[C][43]__[f][66]__[I][49]__[O][4f]_
Orig CT:_[M-^][de]__[M-t][f4]__[M-~][fe]__[M-F][c6]__[f][66]__[M-^I][89]_
Orig Key:_[u][75]__[m][6d]__[C][43]__[f][66]__[I][49]__[O][4f]_
CT 4 verify:_[M-^][de]__[M-t][f4]__[M-~][fe]__[M-F][c6]__[f][66]__[M-^I
   ][89]_
We have Verification also.
Limits:               nednodes=128:ppn=8,nodes=128:ppn=8,walltime=03:00:00
Resources:     cput=904:16:42,mem=265884608kb,vmem=452404156kb,walltime
   =00:53:19

45697582
Lead at t=1 core=1, m=0, key1stB_o_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
```

```
Found CT:_[M-^C][83]__[M-&][a6]__['][27]__[M-^U][95]__[>][3e]__[M-g][e7]_
Recovered Key(EP-1):_[o][6f]__[w][77]__[S][53]__[m][6d]__[6][36]__[j][6a]_
Chain SP:_[o][6f]__[w][77]__[S][53]__[m][6d]__[6][36]__[j][6a]_
Orig CT:_[M-I][c9]__[M-*][aa]__[M-7][b7]__[^C][03]__[M-5][b5]__[ ][09]_
Orig Key:_[o][6f]__[w][77]__[S][53]__[m][6d]__[6][36]__[j][6a]_
CT 4 verify:_[M-I][c9]__[M-*][aa]__[M-7][b7]__[^C][03]__[M-5][b5]__[ ][09]
   _
We have Verification also.
Limits:               neednodes=128:ppn=8,nodes=128:ppn=8,walltime=03:00:00
Resources:     cput=904:04:08,mem=265889284kb,vmem=452404160kb,walltime
   =00:53:18

45697581
Lead at t=1 core=1, m=0, key1stB_D_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[M-q][f1]__[M-/][af]__[M-E][c5]__[0][30]__[[][5b]__[^T][14]_
Recovered Key(EP-1):_[D][44]__[C][43]__[3][33]__[y][79]__[8][38]__[X][58]_
Chain SP:_[D][44]__[C][43]__[3][33]__[y][79]__[8][38]__[X][58]_
Orig CT:_[M-^][de]__[M-%][a5]__[M-+][ab]__[^][5e]__[M-3][b3]__[M-o][ef]_
Orig Key:_[D][44]__[C][43]__[3][33]__[y][79]__[8][38]__[X][58]_
CT 4 verify:_[M-^][de]__[M-%][a5]__[M-+][ab]__[^][5e]__[M-3][b3]__[M-o][ef
   ]_
We have Verification also.
Limits:               neednodes=128:ppn=8,nodes=128:ppn=8,walltime=03:00:00
Resources:     cput=903:57:20,mem=266000072kb,vmem=452404252kb,walltime
   =00:53:20

45697509
][0a]__[M-)][a9]__[M-]][dd]__[M-\][dc]_
We have Verification also.
Limits:               neednodes=128:ppn=8,nodes=128:ppn=8,walltime=01:00:00
Resources:     cput=904:15:11,mem=266171404kb,vmem=452404332kb,walltime
   =00:53:25

45697508
Lead at t=1 core=1, m=0, key1stB_l_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[b][62]__[M-!][a1]__[M-.][ae]__[M-5][b5]__[M-)][a9]__[)][29]_
```

```
Recovered Key(EP-1):_[l][6c]__[W][57]__[e][65]__[K][4b]__[X][58]__[J][4a]_
Chain SP:_[l][6c]__[W][57]__[e][65]__[K][4b]__[X][58]__[J][4a]_
Orig CT:_[M-c][e3]__[M--][ad]__[e][65]__[M->][be]__[^W][17]__[^[][1b]_
Orig Key:_[l][6c]__[W][57]__[e][65]__[K][4b]__[X][58]__[J][4a]_
CT 4 verify:_[M-c][e3]__[M--][ad]__[e][65]__[M->][be]__[^W][17]__[^[][1b]_
We have Verification also.
Limits:               neednodes=128:ppn=8,nodes=128:ppn=8,walltime=01:00:00
Resources:     cput=904:10:34,mem=265995080kb,vmem=452404284kb,walltime
   =00:53:24

45697507
Lead at t=1 core=1, m=0, key1stB_q_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[;][3b]__[M-u][f5]__[M-,][ac]__[^D][04]__[^H][08]__[M-O][cf]_
Recovered Key(EP-1):_[q][71]__[W][57]__[8][38]__[P][50]__[v][76]__[w][77]_
Chain SP:_[q][71]__[W][57]__[8][38]__[P][50]__[v][76]__[w][77]_
Orig CT:_[M-^G][87]__[<][3c]__[M-9][b9]__[M-T][d4]__[M-^^][9e]__[2][32]_
Orig Key:_[q][71]__[W][57]__[8][38]__[P][50]__[v][76]__[w][77]_
CT 4 verify:_[M-^G][87]__[<][3c]__[M-9][b9]__[M-T][d4]__[M-^^][9e]__
   [2][32]_
We have Verification also.
Limits:               neednodes=128:ppn=8,nodes=128:ppn=8,walltime=01:00:00
Resources:     cput=904:04:47,mem=266078880kb,vmem=452404112kb,walltime
   =00:53:27

45697505
Lead at t=1 core=1, m=0, key1stB_F_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[M-^V][96]__[X][58]__[^]][1d]__[^[][1b]__[M-<][bc]__[M-7][b7]_
Recovered Key(EP-1):_[F][46]__[W][57]__[O][4f]__[r][72]__[F][46]__[5][35]_
Chain SP:_[F][46]__[W][57]__[O][4f]__[r][72]__[F][46]__[5][35]_
Orig CT:_[d][64]__[2][32]__[2][32]__[M-^B][82]__[.][2e]__[6][36]_
Orig Key:_[F][46]__[W][57]__[O][4f]__[r][72]__[F][46]__[5][35]_
CT 4 verify:_[d][64]__[2][32]__[2][32]__[M-^B][82]__[.][2e]__[6][36]_
We have Verification also.
Limits:               neednodes=128:ppn=8,nodes=128:ppn=8,walltime=01:00:00
Resources:     cput=904:08:54,mem=266089628kb,vmem=452404296kb,walltime
   =00:53:24
```

```
45697506
Lead at t=1 core=1, m=0, key1stB_k_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[M-x][f8]__[s][73]__()][29]__[^E][05]__[M-Y][d9]__[M-]][dd]_
Recovered Key(EP-1):_[k][6b]__[i][69]__[C][43]__[p][70]__[A][41]__[5][35]_
Chain SP:_[k][6b]__[i][69]__[C][43]__[p][70]__[A][41]__[5][35]_
Orig CT:_[ ][20]__[f][66]__[M-$][a4]__[M-{][fb]__[M-^K][8b]__[M-)][a9]_
Orig Key:_[k][6b]__[i][69]__[C][43]__[p][70]__[A][41]__[5][35]_
CT 4 verify:_[ ][20]__[f][66]__[M-$][a4]__[M-{][fb]__[M-^K][8b]__[M-)][a9]
   _
We have Verification also.
Limits:               nednodes=128:ppn=8,nodes=128:ppn=8,walltime=01:00:00
Resources:     cput=904:11:42,mem=266005628kb,vmem=452404332kb,walltime
   =00:53:22

45697504
Lead at t=1 core=1, m=0, key1stB_c_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[^G][07]__[M-=][bd]__[M-e][e5]__[M-^L][8c]__[M-‘][e0]__[M-^G
   ][87]_
Recovered Key(EP-1):_[c][63]__[3][33]__[z][7a]__[Q][51]__[I][49]__[5][35]_
Chain SP:_[c][63]__[3][33]__[z][7a]__[Q][51]__[I][49]__[5][35]_
Orig CT:_[M-.][ae]__[~][7e]__[M-t][f4]__[N][4e]__[^A][01]__[M-W][d7]_
Orig Key:_[c][63]__[3][33]__[z][7a]__[Q][51]__[I][49]__[5][35]_
CT 4 verify:_[M-.][ae]__[~][7e]__[M-t][f4]__[N][4e]__[^A][01]__[M-W][d7]_
We have Verification also.
Limits:               nednodes=128:ppn=8,nodes=128:ppn=8,walltime=01:00:00
Resources:     cput=904:10:22,mem=265983232kb,vmem=452404240kb,walltime
   =00:53:27

45697503
][0a]_
Recovered Key(EP-1):_[j][6a]__[s][73]__[t][74]__[Q][51]__[y][79]__[7][37]_
Chain SP:_[j][6a]__[s][73]__[t][74]__[Q][51]__[y][79]__[7][37]_
Orig CT:_[^L][0c]__[w][77]__[M-a][e1]__[y][79]__[M-^P][90]__[W][57]_
Orig Key:_[j][6a]__[s][73]__[t][74]__[Q][51]__[y][79]__[7][37]_
CT 4 verify:_[^L][0c]__[w][77]__[M-a][e1]__[y][79]__[M-^P][90]__[W][57]_
```

```
We have Verification also.
Limits:               neednodes=128:ppn=8,nodes=128:ppn=8,walltime=01:00:00
Resources:     cput=904:14:45,mem=266090764kb,vmem=452404220kb,walltime
   =00:53:25

45697502
Lead at t=1 core=1, m=0, key1stB_u_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[h][68]__[M-J][ca]__[M-y][f9]__[M-R][d2]__[:][3a]__[!][21]_
Recovered Key(EP-1):_[u][75]__[L][4c]__[r][72]__[M][4d]__[Y][59]__[l][6c]_
Chain SP:_[u][75]__[L][4c]__[r][72]__[M][4d]__[Y][59]__[l][6c]_
Orig CT:_[k][6b]__[1][31]__[M-0][b0]__[3][33]__[^^][1e]__[(][28]_
Orig Key:_[u][75]__[L][4c]__[r][72]__[M][4d]__[Y][59]__[l][6c]_
CT 4 verify:_[k][6b]__[1][31]__[M-0][b0]__[3][33]__[^^][1e]__[(][28]_
We have Verification also.
Limits:               neednodes=128:ppn=8,nodes=128:ppn=8,walltime=01:00:00
Resources:     cput=904:16:48,mem=266038124kb,vmem=452404240kb,walltime
   =00:53:25

45697501
Lead at t=1 core=1, m=0, key1stB_9_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[|][7c]__[F][46]__[t][74]__[^H][08]__[M-^F][86]__[j][6a]_
Recovered Key(EP-1):_[9][39]__[H][48]__[3][33]__[k][6b]__[Y][59]__[s][73]_
Chain SP:_[9][39]__[H][48]__[3][33]__[k][6b]__[Y][59]__[s][73]_
Orig CT:_[M-F][c6]__[M-^D][84]__[M-?][bf]__[v][76]__[M-1][b1]__[M-7][b7]_
Orig Key:_[9][39]__[H][48]__[3][33]__[k][6b]__[Y][59]__[s][73]_
CT 4 verify:_[M-F][c6]__[M-^D][84]__[M-?][bf]__[v][76]__[M-1][b1]__[M-7][
   b7]_
We have Verification also.
Limits:               neednodes=128:ppn=8,nodes=128:ppn=8,walltime=01:00:00
Resources:     cput=904:15:02,mem=266001912kb,vmem=452404244kb,walltime
   =00:53:26

45697500
Lead at t=1 core=1, m=0, key1stB_r_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[M-U][d5]__[M-^J][8a]__[@][40]__[^?][7f]__[M-5][b5]__[^T][14]_
```

```
Recovered Key(EP-1):_[r][72]__[y][79]__[F][46]__[H][48]__[a][61]__[d][64]_
Chain SP:_[r][72]__[y][79]__[F][46]__[H][48]__[a][61]__[d][64]_
Orig CT:_[W][57]__[M-C][c3]__[V][56]__[M-^U][95]__[M-k][eb]__[M-*][aa]_
Orig Key:_[r][72]__[y][79]__[F][46]__[H][48]__[a][61]__[d][64]_
CT 4 verify:_[W][57]__[M-C][c3]__[V][56]__[M-^U][95]__[M-k][eb]__[M-*][aa]
   _
We have Verification also.
Limits:               nednodes=128:ppn=8,nodes=128:ppn=8,walltime=01:00:00
Resources:     cput=904:16:55,mem=266106168kb,vmem=452404232kb,walltime
   =00:53:28
```

Observation: at a total M of 3,218,079,744 and T of 1024, after dividing total M over 1023 child cores for a 3,145,728 per core we get a finish time of about 53 minutes which means that all runs with 1, 2, 3, or 4 walltime will succeed.

A.1.2 The 1G M records batch

Job ID 45697610 45697609 45697608 45697607 45697606 45697605 45697604 45697603 45697602 45697601 45697568 45697569 45697570 45697567 45697566 45697565 45697562 45697564 45697563 45697561 45697529 45697528 45697527 45697526 45697525 45697524 45697523 45697522 45697521 45697520 45697657 45697656 45697655 45697654 45697653 45697652 45697651 45697650 45697649 45697648 45697488 45697487 45697486 45697485 45697484 45697483 45697482 45697481 45697480 45697479

```
The number of booked cores is 1024 (1 Parent and 1023 children)
The job were successful.
plaintext is: PTPTPT
Random key: (multiple- see below) and it is used as the start point of the
    first chain.
After encrypting the plaintext T times (1024) the chain end point becomes
   : (multiple- see below)
Ciphertext Challenge: (multiple- see below)
CT decrypts back to: (multiple- see below)
output log: (multiple)
Output size (printf overhead): around 297000 lines, 14500000 characters.
error log: (multiple)
Job log: (multiple)
```

```
source code file:(multiple)
job submission script:(multiple)
object file:(multiple)
Cores: 1024
usableCores: 1023
PID:(multiple)
M= 1072693248
C=1024
W=4:00:00
T=1024
B=48
E=2
H=0
L=0
S=1
R=GPC
notes=(multiple)
```

Observation: 1072693248 total M with 1023 cores means 1048576 per core.

```
45697610
Lead at t=1 core=1, m=0, key1stB_q_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[M-y][f9]__[M-f][e6]__[^U][15]__[M- ][a0]__[M-^T][94]__[M-9][b9]
   _
Recovered Key(EP-1):_[q][71]__[5][35]__[a][61]__[N][4e]__[w][77]__[d][64]_
Chain SP:_[q][71]__[5][35]__[a][61]__[N][4e]__[w][77]__[d][64]_
Orig CT:_[=][3d]__[M-x][f8]__[M-:][ba]__[M-;][bb]__[M-/][af]__[M-Z][da]_
Orig Key:_[q][71]__[5][35]__[a][61]__[N][4e]__[w][77]__[d][64]_
CT 4 verify:_[=][3d]__[M-x][f8]__[M-:][ba]__[M-;][bb]__[M-/][af]__[M-Z][da
   ]_
We have Verification also.
Limits:               nednodes=128:ppn=8,nodes=128:ppn=8,walltime=04:00:00
Resources:     cput=305:31:47,mem=207169896kb,vmem=393683984kb,walltime
   =00:18:09

45697609
Lead at t=1 core=1, m=0, key1stB_v_
```

```
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[I][49]__[^@][00]__[^L][0c]__[N][4e]__[M-@][c0]__[M-h][e8]_
Recovered Key(EP-1):_[v][76]__[z][7a]__[r][72]__[9][39]__[7][37]__[d][64]_
Chain SP:_[v][76]__[z][7a]__[r][72]__[9][39]__[7][37]__[d][64]_
Orig CT:_[0][30]__[%][25]__[c][63]__[[][5b]__[M-x][f8]__[R][52]_
Orig Key:_[v][76]__[z][7a]__[r][72]__[9][39]__[7][37]__[d][64]_
CT 4 verify:_[0][30]__[%][25]__[c][63]__[[][5b]__[M-x][f8]__[R][52]_
We have Verification also.
Limits:               nednodes=128:ppn=8,nodes=128:ppn=8,walltime=04:00:00
Resources:     cput=305:32:35,mem=207122956kb,vmem=393684008kb,walltime
   =00:18:10

45697608
Lead at t=1 core=1, m=0, key1stB_Z_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[>][3e]__[U][55]__[V][56]__[M-M][cd]__[M-W][d7]__[.][2e]_
Recovered Key(EP-1):_[Z][5a]__[G][47]__[C][43]__[X][58]__[g][67]__[2][32]_
Chain SP:_[Z][5a]__[G][47]__[C][43]__[X][58]__[g][67]__[2][32]_
Orig CT:_[]][5d]__[J][4a]__[)][29]__[2][32]__[s][73]__[M-N][ce]_
Orig Key:_[Z][5a]__[G][47]__[C][43]__[X][58]__[g][67]__[2][32]_
CT 4 verify:_[]][5d]__[J][4a]__[)][29]__[2][32]__[s][73]__[M-N][ce]_
We have Verification also.
Limits:               nednodes=128:ppn=8,nodes=128:ppn=8,walltime=04:00:00
Resources:     cput=305:33:18,mem=207205896kb,vmem=393684068kb,walltime
   =00:18:08

45697607
Lead at t=1 core=1, m=0, key1stB_q_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[(][28]__[M-^D][84]__[^B][02]__[P][50]__[^C][03]__[M-H][c8]_
Recovered Key(EP-1):_[q][71]__[P][50]__[p][70]__[p][70]__[i][69]__[n][6e]_
Chain SP:_[q][71]__[P][50]__[p][70]__[p][70]__[i][69]__[n][6e]_
Orig CT:_[M-^C][83]__[M-s][f3]__[M-^][de]__[^L][0c]__[M-^D][84]__[M-z][fa]
   _
Orig Key:_[q][71]__[P][50]__[p][70]__[p][70]__[i][69]__[n][6e]_
CT 4 verify:_[M-^C][83]__[M-s][f3]__[M-^][de]__[^L][0c]__[M-^D][84]__[M-z
   ][fa]_
We have Verification also.
```

```
Limits:               neednodes=128:ppn=8,nodes=128:ppn=8,walltime=04:00:00
Resources:     cput=305:33:06,mem=207259332kb,vmem=393684012kb,walltime
   =00:18:09

45697606
Lead at t=1 core=1, m=0, key1stB_3_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[4][34]__[M-]][dd]__[k][6b]__[^B][02]__[M-z][fa]__[M-"][a2]_
Recovered Key(EP-1):_[3][33]__[n][6e]__[b][62]__[n][6e]__[q][71]__[w][77]_
Chain SP:_[3][33]__[n][6e]__[b][62]__[n][6e]__[q][71]__[w][77]_
Orig CT:_[M-V][d6]__[-][2d]__[M-{][fb]__[M-S][d3]__[M-S][d3]__[M-?][bf]_
Orig Key:_[3][33]__[n][6e]__[b][62]__[n][6e]__[q][71]__[w][77]_
CT 4 verify:_[M-V][d6]__[-][2d]__[M-{][fb]__[M-S][d3]__[M-S][d3]__[M-?][bf
   ]_
We have Verification also.
Limits:               neednodes=128:ppn=8,nodes=128:ppn=8,walltime=04:00:00
Resources:     cput=305:34:18,mem=207118652kb,vmem=393683844kb,walltime
   =00:18:10

45697605
Lead at t=1 core=1, m=0, key1stB_Q_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[M-^D][84]__[M-D][c4]__[M-|][fc]__[M-'][a7]__[&][26]__[M-^J][8a]
   _
Recovered Key(EP-1):_[Q][51]__[x][78]__[L][4c]__[c][63]__[p][70]__[8][38]_
Chain SP:_[Q][51]__[x][78]__[L][4c]__[c][63]__[p][70]__[8][38]_
Orig CT:_[r][72]__[M-=][bd]__[]][5d]__[M-Z][da]__[M-!][a1]__[?][3f]_
Orig Key:_[Q][51]__[x][78]__[L][4c]__[c][63]__[p][70]__[8][38]_
CT 4 verify:_[r][72]__[M-=][bd]__[]][5d]__[M-Z][da]__[M-!][a1]__[?][3f]_
We have Verification also.
Limits:               neednodes=128:ppn=8,nodes=128:ppn=8,walltime=04:00:00
Resources:     cput=305:34:04,mem=207151416kb,vmem=393683992kb,walltime
   =00:18:06

45697604
Lead at t=1 core=1, m=0, key1stB_T_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[l][6c]__[M-?][bf]__[M-b][e2]__[^D][04]__[p][70]__[M-^I][89]_
```

```
Recovered Key(EP-1):_[T][54]__[x][78]__[8][38]__[L][4c]__[e][65]__[K][4b]_
Chain SP:_[T][54]__[x][78]__[8][38]__[L][4c]__[e][65]__[K][4b]_
Orig CT:_[M-^K][8b]__['][27]__[q][71]__[K][4b]__[w][77]__[$][24]_
Orig Key:_[T][54]__[x][78]__[8][38]__[L][4c]__[e][65]__[K][4b]_
CT 4 verify:_[M-^K][8b]__['][27]__[q][71]__[K][4b]__[w][77]__[$][24]_
We have Verification also.
Limits:               neednodes=128:ppn=8,nodes=128:ppn=8,walltime=04:00:00
Resources:     cput=305:37:17,mem=207184168kb,vmem=393683944kb,walltime
   =00:18:10

45697603
Lead at t=1 core=1, m=0, key1stB_9_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[T][54]__[^O][0f]__[.][2e]__[M-c][e3]__[M-^X][98]__[M-E][c5]_
Recovered Key(EP-1):_[9][39]__[F][46]__[Q][51]__[g][67]__[Y][59]__[d][64]_
Chain SP:_[9][39]__[F][46]__[Q][51]__[g][67]__[Y][59]__[d][64]_
Orig CT:_[L][4c]__[6][36]__[M-_][df]__[M-v][f6]__[M-^O][8f]__[M-(][a8]_
Orig Key:_[9][39]__[F][46]__[Q][51]__[g][67]__[Y][59]__[d][64]_
CT 4 verify:_[L][4c]__[6][36]__[M-_][df]__[M-v][f6]__[M-^O][8f]__[M-(][a8]
   _
We have Verification also.
Limits:               neednodes=128:ppn=8,nodes=128:ppn=8,walltime=04:00:00
Resources:     cput=305:39:33,mem=207102628kb,vmem=393683968kb,walltime
   =00:18:07

45697602
Lead at t=1 core=1, m=0, key1stB_Y_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[M-9][b9]__[A][41]__[M-^E][85]__[M-H][c8]__[@][40]__[M-^A][81]_
Recovered Key(EP-1):_[Y][59]__[T][54]__[w][77]__[4][34]__[R][52]__[k][6b]_
Chain SP:_[Y][59]__[T][54]__[w][77]__[4][34]__[R][52]__[k][6b]_
Orig CT:_[T][54]__[M-Y][d9]__[4][34]__[M-^]][9d]__[M-x][f8]__[M-^H][88]_
Orig Key:_[Y][59]__[T][54]__[w][77]__[4][34]__[R][52]__[k][6b]_
CT 4 verify:_[T][54]__[M-Y][d9]__[4][34]__[M-^]][9d]__[M-x][f8]__[M-^H
   ][88]_
We have Verification also.
Limits:               neednodes=128:ppn=8,nodes=128:ppn=8,walltime=04:00:00
Resources:     cput=305:34:44,mem=207114624kb,vmem=393684012kb,walltime
```

```
   =00:18:13

45697601
Lead at t=1 core=1, m=0, key1stB_G_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[6][36]__[n][6e]__[A][41]__[M-z][fa]__[M-/][af]__[M-u][f5]_
Recovered Key(EP-1):_[G][47]__[t][74]__[7][37]__[a][61]__[1][31]__[K][4b]_
Chain SP:_[G][47]__[t][74]__[7][37]__[a][61]__[1][31]__[K][4b]_
Orig CT:_[l][6c]__[M-D][c4]__[M-h][e8]__[<][3c]__[M-^B][82]__[M-^S][93]_
Orig Key:_[G][47]__[t][74]__[7][37]__[a][61]__[1][31]__[K][4b]_
CT 4 verify:_[l][6c]__[M-D][c4]__[M-h][e8]__[<][3c]__[M-^B][82]__[M-^S
   ][93]_
We have Verification also.
Limits:               nednodes=128:ppn=8,nodes=128:ppn=8,walltime=04:00:00
Resources:     cput=305:40:20,mem=207079848kb,vmem=393684084kb,walltime
   =00:18:11

45697568
Lead at t=1 core=1, m=0, key1stB_G_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[n][6e]__[M-o][ef]__[+][2b]__[/][2f]__[^^][1e]__[S][53]_
Recovered Key(EP-1):_[G][47]__[7][37]__[o][6f]__[8][38]__[L][4c]__[x][78]_
Chain SP:_[G][47]__[7][37]__[o][6f]__[8][38]__[L][4c]__[x][78]_
Orig CT:_[T][54]__[M-a][e1]__[u][75]__[&][26]__[a][61]__[M-q][f1]_
Orig Key:_[G][47]__[7][37]__[o][6f]__[8][38]__[L][4c]__[x][78]_
CT 4 verify:_[T][54]__[M-a][e1]__[u][75]__[&][26]__[a][61]__[M-q][f1]_
We have Verification also.
Limits:               nednodes=128:ppn=8,nodes=128:ppn=8,walltime=03:00:00
Resources:     cput=306:03:33,mem=207106932kb,vmem=393684016kb,walltime
   =00:18:17

45697569
Lead at t=1 core=1, m=0, key1stB_4_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[{][7b]__[z][7a]__[v][76]__[M-^W][97]__[d][64]__[^B][02]_
Recovered Key(EP-1):_[4][34]__[1][31]__[u][75]__[S][53]__[Y][59]__[Q][51]_
Chain SP:_[4][34]__[1][31]__[u][75]__[S][53]__[Y][59]__[Q][51]_
Orig CT:_[    ][09]__[M-^S][93]__[M-]][dd]__[=][3d]__[M-A][c1]__[M-<][bc]_
```

```
Orig Key:_[4][34]__[1][31]__[u][75]__[S][53]__[Y][59]__[Q][51]_
CT 4 verify:_[ ][09]__[M-^S][93]__[M-]][dd]__[=][3d]__[M-A][c1]__[M-<][bc]
   _
We have Verification also.
Limits:               nednodes=128:ppn=8,nodes=128:ppn=8,walltime=03:00:00
Resources:     cput=305:38:36,mem=207142708kb,vmem=393684052kb,walltime
   =00:18:11

45697570
Lead at t=1 core=1, m=0, key1stB_3_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[q][71]__[T][54]__[\][5c]__[M-I][c9]__[^\][1c]__[^X][18]_
Recovered Key(EP-1):_[3][33]__[4][34]__[h][68]__[n][6e]__[Y][59]__[K][4b]_
Chain SP:_[3][33]__[4][34]__[h][68]__[n][6e]__[Y][59]__[K][4b]_
Orig CT:_[M-^_][9f]__[^G][07]__[3][33]__[M-^_][9f]__[G][47]__[M-j][ea]_
Orig Key:_[3][33]__[4][34]__[h][68]__[n][6e]__[Y][59]__[K][4b]_
CT 4 verify:_[M-^_][9f]__[^G][07]__[3][33]__[M-^_][9f]__[G][47]__[M-j][ea]
   _
We have Verification also.
Limits:               nednodes=128:ppn=8,nodes=128:ppn=8,walltime=03:00:00
Resources:     cput=305:32:22,mem=207127960kb,vmem=393684000kb,walltime
   =00:18:09

45697567
Lead at t=1 core=1, m=0, key1stB_h_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[M-^N][8e]__[M-^K][8b]__[b][62]__[M-4][b4]__[M-^K][8b]__[^^][1e]
   _
Recovered Key(EP-1):_[h][68]__[a][61]__[0][30]__[B][42]__[I][49]__[o][6f]_
Chain SP:_[h][68]__[a][61]__[0][30]__[B][42]__[I][49]__[o][6f]_
Orig CT:_[M-6][b6]__[M-Q][d1]__[M-V][d6]__[M-^E][85]__[M-%][a5]__[M-^Y
   ][99]_
Orig Key:_[h][68]__[a][61]__[0][30]__[B][42]__[I][49]__[o][6f]_
CT 4 verify:_[M-6][b6]__[M-Q][d1]__[M-V][d6]__[M-^E][85]__[M-%][a5]__[M-^Y
   ][99]_
We have Verification also.
Limits:               nednodes=128:ppn=8,nodes=128:ppn=8,walltime=03:00:00
Resources:     cput=305:33:59,mem=207162800kb,vmem=393683968kb,walltime
```

```
   =00:18:11

45697566
Lead at t=1 core=1, m=0, key1stB_2_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[M-&][a6]__[^P][10]__[U][55]__[M-v][f6]__[M-^?][ff]__[-][2d]_
Recovered Key(EP-1):_[2][32]__[M][4d]__[i][69]__[d][64]__[K][4b]__[P][50]_
Chain SP:_[2][32]__[M][4d]__[i][69]__[d][64]__[K][4b]__[P][50]_
Orig CT:_[M-*][aa]__[k][6b]__[%][25]__[P][50]__[M-h][e8]__[M-.][ae]_
Orig Key:_[2][32]__[M][4d]__[i][69]__[d][64]__[K][4b]__[P][50]_
CT 4 verify:_[M-*][aa]__[k][6b]__[%][25]__[P][50]__[M-h][e8]__[M-.][ae]_
We have Verification also.
Limits:               nodes=128:ppn=8,nodes=128:ppn=8,walltime=03:00:00
Resources:     cput=305:35:43,mem=207297040kb,vmem=393683984kb,walltime
   =00:18:24

45697565
Lead at t=1 core=1, m=0, key1stB_c_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[+][2b]__[R][52]__[M-8][b8]__[d][64]__[M-E][c5]__[M-0][b0]_
Recovered Key(EP-1):_[c][63]__[N][4e]__[c][63]__[H][48]__[M][4d]__[e][65]_
Chain SP:_[c][63]__[N][4e]__[c][63]__[H][48]__[M][4d]__[e][65]_
Orig CT:_[M-[][db]__[^O][0f]__[M-+][ab]__[s][73]__[ ][09]__[M-F][c6]_
Orig Key:_[c][63]__[N][4e]__[c][63]__[H][48]__[M][4d]__[e][65]_
CT 4 verify:_[M-[][db]__[^O][0f]__[M-+][ab]__[s][73]__[ ][09]__[M-F][c6]_
We have Verification also.
Limits:               nodes=128:ppn=8,nodes=128:ppn=8,walltime=03:00:00
Resources:     cput=305:48:27,mem=207045144kb,vmem=393684012kb,walltime
   =00:18:07

45697562
Lead at t=1 core=1, m=0, key1stB_f_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[M-V][d6]__[M-a][e1]__[M-^A][81]__[M-.][ae]__[M-g][e7]__[8][38]_
Recovered Key(EP-1):_[f][66]__[D][44]__[d][64]__[5][35]__[p][70]__[v][76]_
Chain SP:_[f][66]__[D][44]__[d][64]__[5][35]__[p][70]__[v][76]_
Orig CT:_[M-%][a5]__[b][62]__[Z][5a]__[?][3f]__[‘][60]__[5][35]_
Orig Key:_[f][66]__[D][44]__[d][64]__[5][35]__[p][70]__[v][76]_
```

```
CT 4 verify:_[M-%][a5]__[b][62]__[Z][5a]__[?][3f]__[‘][60]__[5][35]_
We have Verification also.
Limits:               nednodes=128:ppn=8,nodes=128:ppn=8,walltime=03:00:00
Resources:     cput=305:43:40,mem=207181552kb,vmem=393684008kb,walltime
   =00:18:16

45697564
Lead at t=1 core=1, m=0, key1stB_r_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[M-*][aa]__[^_][1f]__[M-@][c0]__[M-D][c4]__[D][44]__[h][68]_
Recovered Key(EP-1):_[r][72]__[U][55]__[J][4a]__[M][4d]__[v][76]__[j][6a]_
Chain SP:_[r][72]__[U][55]__[J][4a]__[M][4d]__[v][76]__[j][6a]_
Orig CT:_[k][6b]__[T][54]__[M-\][dc]__[M-(][a8]__[^X][18]__[A][41]_
Orig Key:_[r][72]__[U][55]__[J][4a]__[M][4d]__[v][76]__[j][6a]_
CT 4 verify:_[k][6b]__[T][54]__[M-\][dc]__[M-(][a8]__[^X][18]__[A][41]_
We have Verification also.
Limits:               nednodes=128:ppn=8,nodes=128:ppn=8,walltime=03:00:00
Resources:     cput=305:27:33,mem=207414636kb,vmem=393683924kb,walltime
   =00:18:12

45697563
Lead at t=1 core=1, m=0, key1stB_6_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[M-@][c0]__[M-^[][9b]__[M-W][d7]__[^U][15]__[M-M][cd]__[^Z][1a]_
Recovered Key(EP-1):_[6][36]__[C][43]__[r][72]__[I][49]__[2][32]__[f][66]_
Chain SP:_[6][36]__[C][43]__[r][72]__[I][49]__[2][32]__[f][66]_
Orig CT:_[T][54]__[M-(][a8]__[1][31]__[M-d][e4]__[[][5b]__[M-M][cd]_
Orig Key:_[6][36]__[C][43]__[r][72]__[I][49]__[2][32]__[f][66]_
CT 4 verify:_[T][54]__[M-(][a8]__[1][31]__[M-d][e4]__[[][5b]__[M-M][cd]_
We have Verification also.
Limits:               nednodes=128:ppn=8,nodes=128:ppn=8,walltime=03:00:00
Resources:     cput=305:34:42,mem=207176024kb,vmem=393683880kb,walltime
   =00:18:09

45697561
][0a]__[M-^_][9f]__[M-{][fb]__[:][3a]__[M-*][aa]__[M-d][e4]_
Recovered Key(EP-1):_[w][77]__[W][57]__[v][76]__[R][52]__[R][52]__[T][54]_
Chain SP:_[w][77]__[W][57]__[v][76]__[R][52]__[R][52]__[T][54]_
```

```
Orig CT:_[Z][5a]__[M-5][b5]__[M][4d]__[M-S][d3]__[M-/][af]__[,][2c]_
Orig Key:_[w][77]__[W][57]__[v][76]__[R][52]__[R][52]__[T][54]_
CT 4 verify:_[Z][5a]__[M-5][b5]__[M][4d]__[M-S][d3]__[M-/][af]__[,][2c]_
We have Verification also.
Limits:               neednodes=128:ppn=8,nodes=128:ppn=8,walltime=03:00:00
Resources:     cput=305:34:18,mem=207161196kb,vmem=393683964kb,walltime
   =00:18:11

45697529
Lead at t=1 core=1, m=0, key1stB_w_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[2][32]__[H][48]__[M-^W][97]__[M-e][e5]__[M-Z][da]__[M-^?][ff]_
Recovered Key(EP-1):_[w][77]__[F][46]__[l][6c]__[F][46]__[K][4b]__[A][41]_
Chain SP:_[w][77]__[F][46]__[l][6c]__[F][46]__[K][4b]__[A][41]_
Orig CT:_[M-J][ca]__[M-K][cb]__[2][32]__[4][34]__[M-^?][ff]__[M-^K][8b]_
Orig Key:_[w][77]__[F][46]__[l][6c]__[F][46]__[K][4b]__[A][41]_
CT 4 verify:_[M-J][ca]__[M-K][cb]__[2][32]__[4][34]__[M-^?][ff]__[M-^K][8b
   ]_
We have Verification also.
Limits:               neednodes=128:ppn=8,nodes=128:ppn=8,walltime=02:00:00
Resources:     cput=305:40:07,mem=207154436kb,vmem=393683980kb,walltime
   =00:18:10

45697528
Lead at t=1 core=1, m=0, key1stB_q_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[M-e][e5]__[M-q][f1]__[M-^O][8f]__[M-a][e1]__[^Q][11]__[9][39]_
Recovered Key(EP-1):_[q][71]__[O][4f]__[H][48]__[6][36]__[r][72]__[R][52]_
Chain SP:_[q][71]__[O][4f]__[H][48]__[6][36]__[r][72]__[R][52]_
Orig CT:_[R][52]__[I][49]__[ ][20]__[M-^E][85]__[M-^L][8c]__[a][61]_
Orig Key:_[q][71]__[O][4f]__[H][48]__[6][36]__[r][72]__[R][52]_
CT 4 verify:_[R][52]__[I][49]__[ ][20]__[M-^E][85]__[M-^L][8c]__[a][61]_
We have Verification also.
Limits:               neednodes=128:ppn=8,nodes=128:ppn=8,walltime=02:00:00
Resources:     cput=305:40:37,mem=207370996kb,vmem=393684080kb,walltime
   =00:18:10

45697527
```

```
Lead at t=1 core=1, m=0, key1stB_L_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[^R][12]__[8][38]__[f][66]__[M][4d]__[7][37]__[M-/][af]_
Recovered Key(EP-1):_[L][4c]__[b][62]__[M][4d]__[G][47]__[3][33]__[R][52]_
Chain SP:_[L][4c]__[b][62]__[M][4d]__[G][47]__[3][33]__[R][52]_
Orig CT:_[z][7a]__[M-]][dd]__[q][71]__[1][31]__[H][48]__[^U][15]_
Orig Key:_[L][4c]__[b][62]__[M][4d]__[G][47]__[3][33]__[R][52]_
CT 4 verify:_[z][7a]__[M-]][dd]__[q][71]__[1][31]__[H][48]__[^U][15]_
We have Verification also.
Limits:               neednodes=128:ppn=8,nodes=128:ppn=8,walltime=02:00:00
Resources:     cput=305:37:30,mem=207156624kb,vmem=393683904kb,walltime
   =00:18:13

45697526
Lead at t=1 core=1, m=0, key1stB_b_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[^\][1c]__[M-‘][e0]__[v][76]__[M-2][b2]__[^B][02]__[^N][0e]_
Recovered Key(EP-1):_[b][62]__[w][77]__[t][74]__[o][6f]__[k][6b]__[Q][51]_
Chain SP:_[b][62]__[w][77]__[t][74]__[o][6f]__[k][6b]__[Q][51]_
Orig CT:_[M-,][ac]__[M-_][df]__[M-g][e7]__[M-H][c8]__[M-|][fc]__[M-^Y][99]
   _
Orig Key:_[b][62]__[w][77]__[t][74]__[o][6f]__[k][6b]__[Q][51]_
CT 4 verify:_[M-,][ac]__[M-_][df]__[M-g][e7]__[M-H][c8]__[M-|][fc]__[M-^Y
   ][99]_
We have Verification also.
Limits:               neednodes=128:ppn=8,nodes=128:ppn=8,walltime=02:00:00
Resources:     cput=305:38:34,mem=207198544kb,vmem=393683928kb,walltime
   =00:18:09

45697525
Lead at t=1 core=1, m=0, key1stB_L_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[M-^U][95]__[^K][0b]__[M-P][d0]__[D][44]__[M-(][a8]__[ ][09]_
Recovered Key(EP-1):_[L][4c]__[v][76]__[S][53]__[I][49]__[N][4e]__[l][6c]_
Chain SP:_[L][4c]__[v][76]__[S][53]__[I][49]__[N][4e]__[l][6c]_
Orig CT:_[M-m][ed]__[M-~][fe]__[^?][7f]__[M-Z][da]__[M-1][b1]__[M-I][c9]_
Orig Key:_[L][4c]__[v][76]__[S][53]__[I][49]__[N][4e]__[l][6c]_
CT 4 verify:_[M-m][ed]__[M-~][fe]__[^?][7f]__[M-Z][da]__[M-1][b1]__[M-I][
```

```
   c9]_
We have Verification also.
Limits:               nednodes=128:ppn=8,nodes=128:ppn=8,walltime=02:00:00
Resources:     cput=305:42:55,mem=207104676kb,vmem=393683972kb,walltime
   =00:18:09

45697524
Lead at t=1 core=1, m=0, key1stB_i_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[9][39]__[M-K][cb]__[M-^L][8c]__[^O][0f]__[0][30]__[M-u][f5]_
Recovered Key(EP-1):_[i][69]__[D][44]__[g][67]__[Y][59]__[0][30]__[I][49]_
Chain SP:_[i][69]__[D][44]__[g][67]__[Y][59]__[0][30]__[I][49]_
Orig CT:_[t][74]__[M-f][e6]__[M-V][d6]__[^Y][19]__[&][26]__[^T][14]_
Orig Key:_[i][69]__[D][44]__[g][67]__[Y][59]__[0][30]__[I][49]_
CT 4 verify:_[t][74]__[M-f][e6]__[M-V][d6]__[^Y][19]__[&][26]__[^T][14]_
We have Verification also.
Limits:               nednodes=128:ppn=8,nodes=128:ppn=8,walltime=02:00:00
Resources:     cput=305:34:51,mem=207233612kb,vmem=393683908kb,walltime
   =00:18:09

45697523
Lead at t=1 core=1, m=0, key1stB_q_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[M-J][ca]__[M-S][d3]__[M-w][f7]__[G][47]__[Z][5a]__[M-M][cd]_
Recovered Key(EP-1):_[q][71]__[I][49]__[k][6b]__[M][4d]__[4][34]__[9][39]_
Chain SP:_[q][71]__[I][49]__[k][6b]__[M][4d]__[4][34]__[9][39]_
Orig CT:_[L][4c]__[w][77]__[-][2d]__[B][42]__[‘][60]__[M-^^][9e]_
Orig Key:_[q][71]__[I][49]__[k][6b]__[M][4d]__[4][34]__[9][39]_
CT 4 verify:_[L][4c]__[w][77]__[-][2d]__[B][42]__[‘][60]__[M-^^][9e]_
We have Verification also.
Limits:               nednodes=128:ppn=8,nodes=128:ppn=8,walltime=02:00:00
Resources:     cput=305:39:01,mem=207108968kb,vmem=393684076kb,walltime
   =00:18:07

45697522
Lead at t=1 core=1, m=0, key1stB_t_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[M-2][b2]__[M-5][b5]__[M-J][ca]__[_][5f]__[M-^P][90]__[M-7][b7]_
```

```
Recovered Key(EP-1):_[t][74]__[B][42]__[a][61]__[O][4f]__[4][34]__[p][70]_
Chain SP:_[t][74]__[B][42]__[a][61]__[O][4f]__[4][34]__[p][70]_
Orig CT:_[M-0][b0]__[M-:][ba]__[M-~][fe]__[<][3c]__[M-Q][d1]__[M-!][a1]_
Orig Key:_[t][74]__[B][42]__[a][61]__[O][4f]__[4][34]__[p][70]_
CT 4 verify:_[M-0][b0]__[M-:][ba]__[M-~][fe]__[<][3c]__[M-Q][d1]__[M-!][a1
   ]_
We have Verification also.
Limits:               nednodes=128:ppn=8,nodes=128:ppn=8,walltime=02:00:00
Resources:     cput=305:37:24,mem=207186732kb,vmem=393683976kb,walltime
   =00:18:11

45697521
Lead at t=1 core=1, m=0, key1stB_5_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[)][29]__[;][3b]__[Q][51]__[M-x][f8]__[-][2d]__[M-%][a5]_
Recovered Key(EP-1):_[5][35]__[3][33]__[N][4e]__[3][33]__[O][4f]__[B][42]_
Chain SP:_[5][35]__[3][33]__[N][4e]__[3][33]__[O][4f]__[B][42]_
Orig CT:_[T][54]__[;][3b]__[^Z][1a]__[M-o][ef]__[M-^E][85]__[M-7][b7]_
Orig Key:_[5][35]__[3][33]__[N][4e]__[3][33]__[O][4f]__[B][42]_
CT 4 verify:_[T][54]__[;][3b]__[^Z][1a]__[M-o][ef]__[M-^E][85]__[M-7][b7]_
We have Verification also.
Limits:               nednodes=128:ppn=8,nodes=128:ppn=8,walltime=02:00:00
Resources:     cput=305:38:54,mem=207107800kb,vmem=393683944kb,walltime
   =00:18:09

45697520
Lead at t=1 core=1, m=0, key1stB_Z_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[z][7a]__[M-5][b5]__[w][77]__[O][4f]__[^L][0c]__[M-G][c7]_
Recovered Key(EP-1):_[Z][5a]__[H][48]__[5][35]__[3][33]__[s][73]__[9][39]_
Chain SP:_[Z][5a]__[H][48]__[5][35]__[3][33]__[s][73]__[9][39]_
Orig CT:_[M-9][b9]__[M-k][eb]__[M- ][a0]__[^C][03]__[M->][be]__[M-^N][8e]_
Orig Key:_[Z][5a]__[H][48]__[5][35]__[3][33]__[s][73]__[9][39]_
CT 4 verify:_[M-9][b9]__[M-k][eb]__[M- ][a0]__[^C][03]__[M->][be]__[M-^N
   ][8e]_
We have Verification also.
Limits:               nednodes=128:ppn=8,nodes=128:ppn=8,walltime=02:00:00
Resources:     cput=305:36:20,mem=207091284kb,vmem=393683920kb,walltime
```

```
   =00:18:09

45697657
][0a]__[^\][1c]__[^_][1f]__[ ][20]__[~][7e]_
We have Verification also.
Limits:               neednodes=128:ppn=8,nodes=128:ppn=8,walltime=01:00:00
Resources:     cput=305:36:18,mem=207238608kb,vmem=393683968kb,walltime
   =00:18:10
[Identifies script extraction error due to special characters in the
   output.]

45697656
Lead at t=1 core=1, m=0, key1stB_k_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[&][26]__[M-y][f9]__[_][5f]__[M-^_][9f]__[M-(][a8]__[^B][02]_
Recovered Key(EP-1):_[k][6b]__[9][39]__[J][4a]__[O][4f]__[V][56]__[4][34]_
Chain SP:_[k][6b]__[9][39]__[J][4a]__[O][4f]__[V][56]__[4][34]_
Orig CT:_[M-n][ee]__[M-^\][9c]__[M-"][a2]__[p][70]__[M-d][e4]__[M-D][c4]_
Orig Key:_[k][6b]__[9][39]__[J][4a]__[O][4f]__[V][56]__[4][34]_
CT 4 verify:_[M-n][ee]__[M-^\][9c]__[M-"][a2]__[p][70]__[M-d][e4]__[M-D][
   c4]_
We have Verification also.
Limits:               neednodes=128:ppn=8,nodes=128:ppn=8,walltime=01:00:00
Resources:     cput=305:34:52,mem=207120044kb,vmem=393683996kb,walltime
   =00:18:10

45697655
Lead at t=1 core=1, m=0, key1stB_D_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[M-o][ef]__[ ][20]__[M-^X][98]__[M-8][b8]__[^X][18]__[M-.][ae]_
Recovered Key(EP-1):_[D][44]__[X][58]__[t][74]__[p][70]__[Y][59]__[W][57]_
Chain SP:_[D][44]__[X][58]__[t][74]__[p][70]__[Y][59]__[W][57]_
Orig CT:_[l][6c]__[b][62]__[M-^L][8c]__[>][3e]__[S][53]__[M-x][f8]_
Orig Key:_[D][44]__[X][58]__[t][74]__[p][70]__[Y][59]__[W][57]_
CT 4 verify:_[l][6c]__[b][62]__[M-^L][8c]__[>][3e]__[S][53]__[M-x][f8]_
We have Verification also.
Limits:               neednodes=128:ppn=8,nodes=128:ppn=8,walltime=01:00:00
Resources:     cput=305:59:50,mem=206978448kb,vmem=393683932kb,walltime
```

```
   =00:18:09

45697654
Lead at t=1 core=1, m=0, key1stB_Q_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[r][72]__[M-A][c1]__[M-M][cd]__[M-^L][8c]__[L][4c]__[V][56]_
Recovered Key(EP-1):_[Q][51]__[h][68]__[T][54]__[Z][5a]__[K][4b]__[T][54]_
Chain SP:_[Q][51]__[h][68]__[T][54]__[Z][5a]__[K][4b]__[T][54]_
Orig CT:_[M-;][bb]__[*][2a]__[z][7a]__[^Z][1a]__[M-+][ab]__[e][65]_
Orig Key:_[Q][51]__[h][68]__[T][54]__[Z][5a]__[K][4b]__[T][54]_
CT 4 verify:_[M-;][bb]__[*][2a]__[z][7a]__[^Z][1a]__[M-+][ab]__[e][65]_
We have Verification also.
Limits:               neednodes=128:ppn=8,nodes=128:ppn=8,walltime=01:00:00
Resources:     cput=305:34:55,mem=207195044kb,vmem=393684056kb,walltime
   =00:18:10

45697653
Lead at t=1 core=1, m=0, key1stB_m_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[8][38]__[M-^F][86]__[M-5][b5]__[M->][be]__[M-^N][8e]__[M-!][a1]
   _
Recovered Key(EP-1):_[m][6d]__[v][76]__[Z][5a]__[2][32]__[Q][51]__[R][52]_
Chain SP:_[m][6d]__[v][76]__[Z][5a]__[2][32]__[Q][51]__[R][52]_
Orig CT:_[^][5e]__[b][62]__[M-z][fa]__[9][39]__[I][49]__[M-E][c5]_
Orig Key:_[m][6d]__[v][76]__[Z][5a]__[2][32]__[Q][51]__[R][52]_
CT 4 verify:_[^][5e]__[b][62]__[M-z][fa]__[9][39]__[I][49]__[M-E][c5]_
We have Verification also.
Limits:               neednodes=128:ppn=8,nodes=128:ppn=8,walltime=01:00:00
Resources:     cput=305:39:12,mem=207301464kb,vmem=393683920kb,walltime
   =00:18:12

45697652
Lead at t=1 core=1, m=0, key1stB_3_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[M-^^][9e]__[L][4c]__[M-6][b6]__[@][40]__[M-0][b0]__[M-D][c4]_
Recovered Key(EP-1):_[3][33]__[5][35]__[S][53]__[W][57]__[4][34]__[v][76]_
Chain SP:_[3][33]__[5][35]__[S][53]__[W][57]__[4][34]__[v][76]_
Orig CT:_[<][3c]__[c][63]__[w][77]__[q][71]__[F][46]__[^M][0d]_
```

```
Orig Key:_[3][33]__[5][35]__[S][53]__[W][57]__[4][34]__[v][76]_
CT 4 verify:_[<][3c]__[c][63]__[w][77]__[q][71]__[F][46]__[^M][0d]_
We have Verification also.
Limits:               neednodes=128:ppn=8,nodes=128:ppn=8,walltime=01:00:00
Resources:     cput=305:39:19,mem=207134492kb,vmem=393684012kb,walltime
   =00:18:11

45697651
Lead at t=1 core=1, m=0, key1stB_1_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[.][2e]__[^D][04]__[M-[][db]__[M-m][ed]__[M-(][a8]__[M-^B][82]_
Recovered Key(EP-1):_[1][31]__[c][63]__[G][47]__[z][7a]__[x][78]__[C][43]_
Chain SP:_[1][31]__[c][63]__[G][47]__[z][7a]__[x][78]__[C][43]_
Orig CT:_[M-j][ea]__[L][4c]__[T][54]__[,][2c]__[w][77]__[[][5b]_
Orig Key:_[1][31]__[c][63]__[G][47]__[z][7a]__[x][78]__[C][43]_
CT 4 verify:_[M-j][ea]__[L][4c]__[T][54]__[,][2c]__[w][77]__[[][5b]_
We have Verification also.
Limits:               neednodes=128:ppn=8,nodes=128:ppn=8,walltime=01:00:00
Resources:     cput=305:36:11,mem=207268952kb,vmem=393683988kb,walltime
   =00:18:10

45697650
Lead at t=1 core=1, m=0, key1stB_l_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[T][54]__[M-2][b2]__[M-~][fe]__[^W][17]__[]][5d]__[%][25]_
Recovered Key(EP-1):_[l][6c]__[n][6e]__[4][34]__[P][50]__[1][31]__[U][55]_
Chain SP:_[l][6c]__[n][6e]__[4][34]__[P][50]__[1][31]__[U][55]_
Orig CT:_[M-^^][9e]__[M-^G][87]__[M-\][dc]__[M-'][a7]__[M-^N][8e]__[M-^][
   de]_
Orig Key:_[l][6c]__[n][6e]__[4][34]__[P][50]__[1][31]__[U][55]_
CT 4 verify:_[M-^^][9e]__[M-^G][87]__[M-\][dc]__[M-'][a7]__[M-^N][8e]__[M
   -^][de]_
We have Verification also.
Limits:               neednodes=128:ppn=8,nodes=128:ppn=8,walltime=01:00:00
Resources:     cput=305:39:07,mem=207026664kb,vmem=393684060kb,walltime
   =00:18:06

45697649
```

```
Lead at t=1 core=1, m=0, key1stB_g_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[#][23]__[M-^L][8c]__[M-q][f1]__[M-e][e5]__[^@][00]__[M-^C][83]_
Recovered Key(EP-1):_[g][67]__[9][39]__[U][55]__[s][73]__[s][73]__[u][75]_
Chain SP:_[g][67]__[9][39]__[U][55]__[s][73]__[s][73]__[u][75]_
Orig CT:_[M-^][de]__[l][6c]__[M--][ad]__[^L][0c]__[M-)][a9]__[[][5b]_
Orig Key:_[g][67]__[9][39]__[U][55]__[s][73]__[s][73]__[u][75]_
CT 4 verify:_[M-^][de]__[l][6c]__[M--][ad]__[^L][0c]__[M-)][a9]__[[][5b]_
We have Verification also.
Limits:               nednodes=128:ppn=8,nodes=128:ppn=8,walltime=01:00:00
Resources:     cput=305:37:50,mem=207098552kb,vmem=393684028kb,walltime
   =00:18:13

45697648
Lead at t=1 core=1, m=0, key1stB_n_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[^G][07]__[M-o][ef]__[M-x][f8]__[o][6f]__[^A][01]__[M-C][c3]_
Recovered Key(EP-1):_[n][6e]__[M][4d]__[a][61]__[k][6b]__[D][44]__[2][32]_
Chain SP:_[n][6e]__[M][4d]__[a][61]__[k][6b]__[D][44]__[2][32]_
Orig CT:_[    ][09]__[M-\][dc]__[i][69]__[M-@][c0]__[.][2e]__[M-L][cc]_
Orig Key:_[n][6e]__[M][4d]__[a][61]__[k][6b]__[D][44]__[2][32]_
CT 4 verify:_[ ][09]__[M-\][dc]__[i][69]__[M-@][c0]__[.][2e]__[M-L][cc]_
We have Verification also.
Limits:               nednodes=128:ppn=8,nodes=128:ppn=8,walltime=01:00:00
Resources:     cput=305:36:19,mem=207091740kb,vmem=393684024kb,walltime
   =00:18:08

45697488
Lead at t=1 core=1, m=0, key1stB_M_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[M-^W][97]__[J][4a]__[M-^@][80]__[O][4f]__[M-^M][8d]__[M-6][b6]_
Recovered Key(EP-1):_[M][4d]__[r][72]__[S][53]__[C][43]__[A][41]__[R][52]_
Chain SP:_[M][4d]__[r][72]__[S][53]__[C][43]__[A][41]__[R][52]_
Orig CT:_[M-@][c0]__[M-]][dd]__[M-N][ce]__[M-|][fc]__[^?][7f]__[h][68]_
Orig Key:_[M][4d]__[r][72]__[S][53]__[C][43]__[A][41]__[R][52]_
CT 4 verify:_[M-@][c0]__[M-]][dd]__[M-N][ce]__[M-|][fc]__[^?][7f]__[h][68]
   _
We have Verification also.
```

```
Limits:               neednodes=128:ppn=8,nodes=128:ppn=8,walltime=01:00:00
Resources:     cput=305:41:06,mem=207370804kb,vmem=393683892kb,walltime
   =00:18:10

45697487
Lead at t=1 core=1, m=0, key1stB_g_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[^?][7f]__[^Y][19]__[M-G][c7]__[O][4f]__[t][74]__[%][25]_
Recovered Key(EP-1):_[g][67]__[X][58]__[K][4b]__[o][6f]__[Z][5a]__[4][34]_
Chain SP:_[g][67]__[X][58]__[K][4b]__[o][6f]__[Z][5a]__[4][34]_
Orig CT:_[^L][0c]__[1][31]__[?][3f]__[^Z][1a]__[e][65]__[‘][60]_
Orig Key:_[g][67]__[X][58]__[K][4b]__[o][6f]__[Z][5a]__[4][34]_
CT 4 verify:_[^L][0c]__[1][31]__[?][3f]__[^Z][1a]__[e][65]__[‘][60]_
We have Verification also.
Limits:               neednodes=128:ppn=8,nodes=128:ppn=8,walltime=01:00:00
Resources:     cput=305:51:59,mem=207213224kb,vmem=393684060kb,walltime
   =00:18:08

45697486
Lead at t=1 core=1, m=0, key1stB_t_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[M-l][ec]__[M-m][ed]__[^Z][1a]__[C][43]__[M-Q][d1]__[M-m][ed]_
Recovered Key(EP-1):_[t][74]__[Z][5a]__[n][6e]__[Q][51]__[g][67]__[5][35]_
Chain SP:_[t][74]__[Z][5a]__[n][6e]__[Q][51]__[g][67]__[5][35]_
Orig CT:_[L][4c]__[M-^G][87]__[M-g][e7]__[3][33]__[F][46]__[^Y][19]_
Orig Key:_[t][74]__[Z][5a]__[n][6e]__[Q][51]__[g][67]__[5][35]_
CT 4 verify:_[L][4c]__[M-^G][87]__[M-g][e7]__[3][33]__[F][46]__[^Y][19]_
We have Verification also.
Limits:               neednodes=128:ppn=8,nodes=128:ppn=8,walltime=01:00:00
Resources:     cput=305:47:07,mem=207199860kb,vmem=393684028kb,walltime
   =00:18:10

45697485
][0a]_
Recovered Key(EP-1):_[n][6e]__[I][49]__[p][70]__[E][45]__[X][58]__[N][4e]_
Chain SP:_[n][6e]__[I][49]__[p][70]__[E][45]__[X][58]__[N][4e]_
Orig CT:_[u][75]__[W][57]__[M-^Y][99]__[>][3e]__[]][5d]__[B][42]_
Orig Key:_[n][6e]__[I][49]__[p][70]__[E][45]__[X][58]__[N][4e]_
```

```
CT 4 verify:_[u][75]__[W][57]__[M-^Y][99]__[>][3e]__[]][5d]__[B][42]_
We have Verification also.
Limits:               nednodes=128:ppn=8,nodes=128:ppn=8,walltime=01:00:00
Resources:     cput=305:48:51,mem=207213020kb,vmem=393683996kb,walltime
   =00:18:10
[Identified script extraction error]

45697484
Lead at t=1 core=1, m=0, key1stB_c_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[S][53]__[M-7][b7]__[T][54]__[M-c][e3]__[h][68]__[:][3a]_
Recovered Key(EP-1):_[c][63]__[v][76]__[6][36]__[V][56]__[0][30]__[9][39]_
Chain SP:_[c][63]__[v][76]__[6][36]__[V][56]__[0][30]__[9][39]_
Orig CT:_[M-f][e6]__[w][77]__[‘][60]__[M-t][f4]__[i][69]__[B][42]_
Orig Key:_[c][63]__[v][76]__[6][36]__[V][56]__[0][30]__[9][39]_
CT 4 verify:_[M-f][e6]__[w][77]__[‘][60]__[M-t][f4]__[i][69]__[B][42]_
We have Verification also.
Limits: nednodes=128:ppn=8,nodes=128:ppn=8,walltime=01:00:00
Resources:     cput=305:40:32,mem=207139408kb,vmem=393683872kb,walltime
   =00:18:08

45697483
Lead at t=1 core=1, m=0, key1stB_J_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[@][40]__[M-8][b8]__[9][39]__[M-Q][d1]__[M-^Y][99]__[M-^P][90]_
Recovered Key(EP-1):_[J][4a]__[T][54]__[i][69]__[O][4f]__[I][49]__[I][49]_
Chain SP:_[J][4a]__[T][54]__[i][69]__[O][4f]__[I][49]__[I][49]_
Orig CT:_[M-i][e9]__[M-*][aa]__[M-^L][8c]__[M-,][ac]__[M-I][c9]__[3][33]_
Orig Key:_[J][4a]__[T][54]__[i][69]__[O][4f]__[I][49]__[I][49]_
CT 4 verify:_[M-i][e9]__[M-*][aa]__[M-^L][8c]__[M-,][ac]__[M-I][c9]__
   [3][33]_
We have Verification also.
Limits: nednodes=128:ppn=8,nodes=128:ppn=8,walltime=01:00:00
Resources:     cput=305:46:58,mem=207254808kb,vmem=393684072kb,walltime
   =00:18:09

45697482
Lead at t=1 core=1, m=0, key1stB_6_
```

```
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[t][74]__[M-f][e6]__[g][67]__[M-A][c1]__[M-^D][84]__[M-#][a3]_
Recovered Key(EP-1):_[6][36]__[O][4f]__[e][65]__[8][38]__[p][70]__[P][50]_
Chain SP:_[6][36]__[O][4f]__[e][65]__[8][38]__[p][70]__[P][50]_
Orig CT:_[M-^I][89]__[M-q][f1]__[M-^\][9c]__[u][75]__[|][7c]__[^X][18]_
Orig Key:_[6][36]__[O][4f]__[e][65]__[8][38]__[p][70]__[P][50]_
CT 4 verify:_[M-^I][89]__[M-q][f1]__[M-^\][9c]__[u][75]__[|][7c]__[^X][18]
   _
We have Verification also.
Limits: nednodes=128:ppn=8,nodes=128:ppn=8,walltime=01:00:00
Resources:     cput=305:43:56,mem=207109196kb,vmem=393683992kb,walltime
   =00:18:11

45697481
Lead at t=1 core=1, m=0, key1stB_7_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[M-6][b6]__[^?][7f]__[N][4e]__[M-^\][9c]__[^^][1e]__[M-^O][8f]_
Recovered Key(EP-1):_[7][37]__[j][6a]__[A][41]__[4][34]__[t][74]__[3][33]_
Chain SP:_[7][37]__[j][6a]__[A][41]__[4][34]__[t][74]__[3][33]_
Orig CT:_[M-e][e5]__[x][78]__[e][65]__[M-P][d0]__[M-^R][92]__[M-v][f6]_
Orig Key:_[7][37]__[j][6a]__[A][41]__[4][34]__[t][74]__[3][33]_
CT 4 verify:_[M-e][e5]__[x][78]__[e][65]__[M-P][d0]__[M-^R][92]__[M-v][f6]
   _
We have Verification also.
Limits: nednodes=128:ppn=8,nodes=128:ppn=8,walltime=01:00:00
Resources:     cput=305:42:59,mem=207219704kb,vmem=393684016kb,walltime
   =00:18:08

45697480
Lead at t=1 core=1, m=0, key1stB_Q_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[M-!][a1]__[^B][02]__[M-x][f8]__[^@][00]__[}][7d]__[^\][1c]_
Recovered Key(EP-1):_[Q][51]__[I][49]__[f][66]__[5][35]__[h][68]__[I][49]_
Chain SP:_[Q][51]__[I][49]__[f][66]__[5][35]__[h][68]__[I][49]_
Orig CT:_[#][23]__[^][5e]__[^H][08]__[M-]][dd]__[M-(][a8]__[s][73]_
Orig Key:_[Q][51]__[I][49]__[f][66]__[5][35]__[h][68]__[I][49]_
CT 4 verify:_[#][23]__[^][5e]__[^H][08]__[M-]][dd]__[M-(][a8]__[s][73]_
We have Verification also.
```

```
Limits: neednodes=128:ppn=8,nodes=128:ppn=8,walltime=01:00:00
Resources:     cput=305:48:48,mem=207309956kb,vmem=393683976kb,walltime
   =00:18:13

45697479
Lead at t=1 core=1, m=0, key1stB_S_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[M-V][d6]__[^O][0f]__[m][6d]__[M-Q][d1]__[G][47]__[M-8][b8]_
Recovered Key(EP-1):_[S][53]__[n][6e]__[0][30]__[H][48]__[w][77]__[h][68]_
Chain SP:_[S][53]__[n][6e]__[0][30]__[H][48]__[w][77]__[h][68]_
Orig CT:_[M-4][b4]__[A][41]__[M-w][f7]__[M-^E][85]__[M-/][af]__[-][2d]_
Orig Key:_[S][53]__[n][6e]__[0][30]__[H][48]__[w][77]__[h][68]_
CT 4 verify:_[M-4][b4]__[A][41]__[M-w][f7]__[M-^E][85]__[M-/][af]__[-][2d]
   _
We have Verification also.
Limits: neednodes=128:ppn=8,nodes=128:ppn=8,walltime=01:00:00
Resources:     cput=305:43:36,mem=207258020kb,vmem=393683960kb,walltime
   =00:18:10
```

A.1.3 The 4G M records faulty batch

Job ID 45697516

```
The number of booked cores is 1024 (1 Parent and 1023 children)
The job was killed for exceeding the requested walltime (3630 exceeded
   limit 3600).
plaintext is: PTPTPT
Random key: [0_6e_n][1_55_U][2_42_B][3_75_u][4_43_C][5_61_a] and it is
   used as the start point of the first chain.
After encrypting the plaintext T times (1024) the chain end point becomes
   ___[e4]___[fe]___[73]___[99]___[1b]___[0c]
Ciphertext Challenge: [0_da_][1_4f_O][2 _be_ ][3 _b1_ ][4_97_~W][5_7e_~]
CT decrypts back to: [0_50_P][1_54_T][2_50_P][3_54_T][4_50_P][5_54_T]
Output log: tmto1024t1024c4290772992m1hrs_bestCvrg0.o45697516
Output size (printf overhead): 56288 lines, 4351676 characters.
error log: tmto1024t1024c4290772992m1hrs_bestCvrg0.e45697516
```

```
Job log: tmt_Fri_Oct_20_21_18_09_EDT_2017_M=4290772992_C=1024_W=1_00_00_T
   =1024_B=48_E=2_H=0_L=0_S=1_R=GPC__notes=
   tmto1024t1024c4290772992m1hrs_bestCvrg0_45697516.gpc-sched-ib0.joblog
source code file: tmt_Fri_Oct_20_21_18_09_EDT_2017_M=4290772992_C=1024_W=1
   _00_00_T=1024_B=48_E=2_H=0_L=0_S=1_R=GPC__notes=
   tmto1024t1024c4290772992m1hrs_bestCvrg0.c
job submission script: tmt_Fri_Oct_20_21_18_09_EDT_2017_M=4290772992_C
   =1024_W=1_00_00_T=1024_B=48_E=2_H=0_L=0_S=1_R=GPC__notes=
   tmto1024t1024c4290772992m1hrs_bestCvrg0.sh
object file: tmt_Fri_Oct_20_21_18_09_EDT_2017_M=4290772992_C=1024_W=1
   _00_00_T=1024_B=48_E=2_H=0_L=0_S=1_R=GPC__notes=
   tmto1024t1024c4290772992m1hrs_bestCvrg0.out
Cores:1024
usableCores:1023
PID:5434
Date: tmt_Fri_Oct_20_21_18_09_EDT_2017_
M=4290772992
C=1024
W=1:00:00
T=1024
B=48
E=2
H=0
L=0
S=1
R=GPC
notes=tmto1024t1024c4290772992m1hrs_bestCvrg0.out
```

Observations: The walltime that was booked by the probing script and provided by the GPC was not enough to finish. The job was kicked out while calculating the tables. The overhead of the printing is large and the over all performance can be enhanced by decreasing the output overhead. Similar jobs o45697519, o45697518, o45697517, o45697516, o45697515, o45697513, o45697514, o45697511, o45697510, o45697512 behave similarly. This means it is ot a one time problem but rather with the same environment variables it will respond similarly.

```
e45697510:=>> PBS: job killed: walltime 3622 exceeded limit 3600
```

```
e45697511:=>> PBS: job killed: walltime 3630 exceeded limit 3600
e45697512:=>> PBS: job killed: walltime 3637 exceeded limit 3600
e45697513:=>> PBS: job killed: walltime 3630 exceeded limit 3600
e45697514:=>> PBS: job killed: walltime 3609 exceeded limit 3600
e45697515:=>> PBS: job killed: walltime 3634 exceeded limit 3600
e45697516:=>> PBS: job killed: walltime 3630 exceeded limit 3600
e45697517:=>> PBS: job killed: walltime 3642 exceeded limit 3600
e45697518:=>> PBS: job killed: walltime 3629 exceeded limit 3600
e45697519:=>> PBS: job killed: walltime 3612 exceeded limit 3600
```

Job ID 45697515 is one of them and looking at the details they are more or less the same except the random key is different

```
The number of booked cores is 1024 (1 Parent and 1023 children)
The job was killed for exceeding the requested walltime (3634 exceeded
   limit 3600).
plaintext is: PTPTPT
Random key: [0_65_e][1_55_U][2_51_Q][3_67_g][4_6b_k][5_64_d] and it is
   used as the start point of the first chain.
After encrypting the plaintext T times (1024) the chain end point becomes
   : ___[a1]___[57]___[54]___[3d]___[d6]___[a4]
Ciphertext Challenge: [0_d2_][1 _a2_ ][2_77_w][3_4d_M][4_7e_~][5_c4_]
CT decrypts back to: [0_50_P][1_54_T][2_50_P][3_54_T][4_50_P][5_54_T] so
   we know that there is nothing wrong with the crypto system.
output log: tmto1024t1024c4290772992m1hrs_bestCvrg0.o45697515
Output size (printf overhead): .... lines, ... characters.
error log: tmto1024t1024c4290772992m1hrs_bestCvrg0.e45697515
Job log: tmt_Fri_Oct_20_21_18_08_EDT_2017_M=4290772992_C=1024_W=1_00_00_T
   =1024_B=48_E=2_H=0_L=0_S=1_R=GPC__notes=
   tmto1024t1024c4290772992m1hrs_bestCvrg0_45697515.gpc-sched-ib0.joblog
source code file: tmt_Fri_Oct_20_21_18_08_EDT_2017_M=4290772992_C=1024_W=1
   _00_00_T=1024_B=48_E=2_H=0_L=0_S=1_R=GPC__notes=
   tmto1024t1024c4290772992m1hrs_bestCvrg0.c
job submission script: tmt_Fri_Oct_20_21_18_08_EDT_2017_M=4290772992_C
   =1024_W=1_00_00_T=1024_B=48_E=2_H=0_L=0_S=1_R=GPC__notes=
   tmto1024t1024c4290772992m1hrs_bestCvrg0.sh
object file: tmt_Fri_Oct_20_21_18_08_EDT_2017_M=4290772992_C=1024_W=1
```

```
   _00_00_T=1024_B=48_E=2_H=0_L=0_S=1_R=GPC__notes=
   tmto1024t1024c4290772992m1hrs_bestCvrg0.out
Cores: 1024
usableCores: 1023
PID:1903
Date: tmt_Fri_Oct_20_21_18_08_EDT_2017
M= 4290772992
C=1024
W=1:00:00 hours
T=1024
B=48 bits block size
E= 2= Simeck
H=0
L=0
S= 1= no sorting
R= GPC
notes=tmto1024t1024c4290772992m1hrs_bestCvrg0
```

Job ID 45697550

```
The number of booked cores is 1024 (1 Parent and 1023 children)
plaintext is: PTPTPT
Random key: [0_38_8][1_59_Y][2_52_R][3_48_H][4_78_x][5_62_b] and it is
   used as the start point of the first chain.
After encrypting the plaintext T times (1024), the chain end point becomes
    : ___[0c]___[73]___[6a]___[3a]___[93]___[c4]
Ciphertext Challenge: [0_ba_][1 _a2_ ][2 _bf_ ][3 _ec_ ][4 _ab_ ][5_7e_~]
CT decrypts back to: [0_50_P][1_54_T][2_50_P][3_54_T][4_50_P][5_54_T]
output log: tmto1024t1024c4290772992m2hrs_bestCvrg0.o45697550
Output size (printf overhead): 64586 lines and 4962731 characters.
Error log: tmto1024t1024c4290772992m2hrs_bestCvrg0.e45697550
Error log looks fine. Nothing than the depcation error "Using mlock
   ulimits for SHM_HUGETLB is deprecated".
Job log: tmt_Fri_Oct_20_21_18_46_EDT_2017_M=4290772992_C=1024_W=2_00_00_T
   =1024_B=48_E=2_H=0_L=0_S=1_R=GPC__notes=
   tmto1024t1024c4290772992m2hrs_bestCvrg0_45697550.gpc-sched-ib0.joblog
source code file: tmt_Fri_Oct_20_21_18_46_EDT_2017_M=4290772992_C=1024_W=2
```

```
   _00_00_T=1024_B=48_E=2_H=0_L=0_S=1_R=GPC__notes=
   tmto1024t1024c4290772992m2hrs_bestCvrg0.c
job submission script: tmt_Fri_Oct_20_21_18_46_EDT_2017_M=4290772992_C
   =1024_W=2_00_00_T=1024_B=48_E=2_H=0_L=0_S=1_R=GPC__notes=
   tmto1024t1024c4290772992m2hrs_bestCvrg0.sh
object file: tmt_Fri_Oct_20_21_18_46_EDT_2017_M=4290772992_C=1024_W=2
   _00_00_T=1024_B=48_E=2_H=0_L=0_S=1_R=GPC__notes=
   tmto1024t1024c4290772992m2hrs_bestCvrg0.out
Cores: 1024
usable Cores: 1023
PID: 31480
Date:
M= 4290772992
C= 1024
W= walltime is 2 hours
used walltime: =01:10:29
T= 1024
B= 48
E= 2 Simeck
H= 0
L= 0
S= 1
R= GPC
notes=tmto1024t1024c4290772992m2hrs_bestCvrg0
```

Observation: The framework failed to find the key even after successfully finishing the search. This could be due to a bug. The generation and negative-result search took about 1 hour 10 minute and was in the scope of the 2 hours walltime booked from the GPC system. The following jobs also exhibit and confirm the same behavior:

```
45697550:Limits:      nednodes=128:ppn=8,nodes=128:ppn=8,walltime=02:00:00
45697550:Resources:   cput=1196:46:41,mem=295264104kb,vmem=481764252kb,
   walltime=01:10:29
45697551:Limits:      nednodes=128:ppn=8,nodes=128:ppn=8,walltime=02:00:00
45697551:Resources:   cput=1196:40:14,mem=294980032kb,vmem=481764400kb,
   walltime=01:10:27
45697552:Limits:      nednodes=128:ppn=8,nodes=128:ppn=8,walltime=02:00:00
45697552:Resources:   cput=1196:46:53,mem=295089584kb,vmem=481764344kb,
```

```
   walltime=01:10:33
45697553:Limits:      neednodes=128:ppn=8,nodes=128:ppn=8,walltime=02:00:00
45697553:Resources:   cput=1196:43:44,mem=295023060kb,vmem=481764380kb,
   walltime=01:10:28
45697554:Limits:      neednodes=128:ppn=8,nodes=128:ppn=8,walltime=02:00:00
45697554:Resources:   cput=1196:38:28,mem=295047544kb,vmem=481764396kb,
   walltime=01:10:26
45697555:Limits:      neednodes=128:ppn=8,nodes=128:ppn=8,walltime=02:00:00
45697555:Resources:   cput=1196:42:55,mem=295203680kb,vmem=481764396kb,
   walltime=01:10:32
45697556:Limits:      neednodes=128:ppn=8,nodes=128:ppn=8,walltime=02:00:00
45697556:Resources:   cput=1196:52:04,mem=295036712kb,vmem=481764400kb,
   walltime=01:10:28
45697557:Limits:      neednodes=128:ppn=8,nodes=128:ppn=8,walltime=02:00:00
45697557:Resources:   cput=1196:54:36,mem=295050420kb,vmem=481764216kb,
   walltime=01:10:30
45697559:Limits:      neednodes=128:ppn=8,nodes=128:ppn=8,walltime=02:00:00
45697559:Resources:   cput=1197:00:31,mem=295218280kb,vmem=481764268kb,
   walltime=01:10:30
```

Since there seemed to a pattern we tried to group the runs into cluster. This cluster with the faulty results can be extended to cover similar runs with higher walltime. The fact that they fail to reach the results but finish the run gracefully means that nothing is going to change if the run has more walltime. To test this we look at the the following jobs (45697640 45697639 45697637 45697636 45697638 45697635 45697634 45697633 45697632 45697631 45697600 45697599 45697598 45697594 45697597 45697596 45697595 45697592 45697593 45697591).

```
45697640
e45697640:=>> PBS: job killed: walltime 14422 exceeded limit 14400
o45697640:Limits: neednodes=128:ppn=8,nodes=128:ppn=8,walltime=04:00:00
o45697640:Resources:cput=1183:38:46,mem=295194716kb,vmem=481764380kb,
   walltime=04:00:24

45697639
o45697639:Limits: neednodes=128:ppn=8,nodes=128:ppn=8,walltime=04:00:00
o45697639:Resources:cput=1197:10:17,mem=295178684kb,vmem=481764400kb,
   walltime=01:10:37
```

```
45697637
o45697637:Limits: neednodes=128:ppn=8,nodes=128:ppn=8,walltime=04:00:00
o45697637:Resources:cput=1197:17:55,mem=295176560kb,vmem=481764388kb,
   walltime=01:10:58

45697636
o45697636:Limits: neednodes=128:ppn=8,nodes=128:ppn=8,walltime=04:00:00
o45697636:Resources:cput=1197:10:08,mem=295288984kb,vmem=481764340kb,
   walltime=01:10:48

45697638
o45697638:Limits: neednodes=128:ppn=8,nodes=128:ppn=8,walltime=04:00:00
o45697638:Resources:cput=1197:21:48,mem=295278704kb,vmem=481764412kb,
   walltime=01:10:44

45697635
o45697635:Limits: neednodes=128:ppn=8,nodes=128:ppn=8,walltime=04:00:00
o45697635:Resources:cput=1197:21:47,mem=295416024kb,vmem=481764152kb,
   walltime=01:10:42

45697634
o45697634:Limits: neednodes=128:ppn=8,nodes=128:ppn=8,walltime=04:00:00
o45697634:Resources:cput=1197:14:30,mem=295118820kb,vmem=481764340kb,
   walltime=01:10:34

45697633
o45697633:Limits: neednodes=128:ppn=8,nodes=128:ppn=8,walltime=04:00:00
o45697633:Resources:cput=1197:13:54,mem=295070864kb,vmem=481764368kb,
   walltime=01:10:39

45697632
o45697632:Limits: neednodes=128:ppn=8,nodes=128:ppn=8,walltime=04:00:00
o45697632:Resources:cput=1196:56:47,mem=295065932kb,vmem=481764368kb,
   walltime=01:10:34

45697631
o45697631:Limits: neednodes=128:ppn=8,nodes=128:ppn=8,walltime=04:00:00
```

```
o45697631:Resources:cput=1197:08:27,mem=295094196kb,vmem=481764408kb,
   walltime=01:10:34

45697600
o45697600:Limits: neednodes=128:ppn=8,nodes=128:ppn=8,walltime=03:00:00
o45697600:Resources:cput=1197:02:35,mem=295026248kb,vmem=481764444kb,
   walltime=01:10:31

45697599
o45697599:Limits:    neednodes=128:ppn=8,nodes=128:ppn=8,walltime=03:00:00
o45697599:Resources: cput=1197:07:23,mem=295133240kb,vmem=481764364kb,
   walltime=01:10:29

45697598
o45697598:Limits:    neednodes=128:ppn=8,nodes=128:ppn=8,walltime=03:00:00
o45697598:Resources: cput=1196:54:53,mem=295207340kb,vmem=481764488kb,
   walltime=01:10:33

45697594
e45697594:=>> PBS: job killed: walltime 10806 exceeded limit 10800
o45697594:Limits:    neednodes=128:ppn=8,nodes=128:ppn=8,walltime=03:00:00
o45697594:Resources: cput=1196:43:34,mem=295526676kb,vmem=481764408kb,
   walltime=03:00:08

45697597
o45697597:Limits:    neednodes=128:ppn=8,nodes=128:ppn=8,walltime=03:00:00
o45697597:Resources: cput=1197:20:37,mem=295076616kb,vmem=481764288kb,
   walltime=01:10:33

45697596
o45697596:Limits:    neednodes=128:ppn=8,nodes=128:ppn=8,walltime=03:00:00
o45697596:Resources: cput=1197:01:38,mem=295129672kb,vmem=481764372kb,
   walltime=01:10:31

45697595
o45697595:Limits:    neednodes=128:ppn=8,nodes=128:ppn=8,walltime=03:00:00
o45697595:Resources: cput=1196:54:04,mem=295206048kb,vmem=481764352kb,
   walltime=01:10:32
```

```
45697592
o45697592:Limits:     neednodes=128:ppn=8,nodes=128:ppn=8,walltime=03:00:00
o45697592:Resources:  cput=1197:03:16,mem=295065124kb,vmem=481764328kb,
   walltime=01:10:32

45697593
o45697593:Limits:     neednodes=128:ppn=8,nodes=128:ppn=8,walltime=03:00:00
o45697593:Resources:  cput=1197:10:20,mem=295087108kb,vmem=481764360kb,
   walltime=01:10:33

45697591
o45697591:Limits:     neednodes=128:ppn=8,nodes=128:ppn=8,walltime=03:00:00
o45697591:Resources:  cput=1196:56:45,mem=295036804kb,vmem=481764376kb,
   walltime=01:10:30
```

The behavior is consistent except for job 45697640. Considering the whole cluster of jobs, they all finishes in about 1:10 hours if they had more than one hour of walltime booked. If the job had just one hour booked it it safe to assume that it was killed just around 10 minutes before it was finished. Job 45697640 stands out as an outlier and is an opportunity for more investigation.

A.1.4 The 15Mega M batch

Using the same clustering approach we can analyze the following set of jobs together (o45697767 o45697766 o45697765 o45697762 o45697764 o45697763 o45697761 o45697759 o45697760 o45697758 o45697735 o45697737 o45697736 o45697733 o45697734 o45697732 o45697731 o45697730 o45697728 o45697729 o45697706 o45697707 o45697704 o45697705 o45697703 o45697702 o45697701 o45697700 o45697699 o45697698 o45697676 o45697677 o45697672 o45697673 o45697671 o45697674 o45697675 o45697668 o45697670 o45697669). All these jobs have the same parameter except for the random key and the requested walltime.

Job ID (multiple)

```
The number of booked cores is 1024 (1 Parent and 1023 children)
All jobs achieved the whole cycle. They were able to initiate, distribute,
    caluclate tables and find the random key.
```

```
plaintext is: PTPTPT
Random key: (multiple)
After encrypting the plaintext T times (1024) the chain end point becomes
   : (multiple)
Ciphertext Challenge:(multiple)
CT decrypts back to: PTPTPT
output log: (multiple)
Output size (printf overhead): in the vicinity of 1200 lines, 550,000
   characters.
error log: (multiple)
Job log: (multiple)
source code file:(multiple)
job submission script:(multiple)
object file:(multiple)
Cores: 1024
usable Cores:1023
PID: (multiple)
Date: (multiple)
M= 15728640
C= 1024
W= 1, 2, 3, and 4 hours
T= 1024
B= 48
E= 2 Simeck
H=0
L=0
S=1
R=GPC
notes= (Multiple)
```

Observation: Looking at the following runs they all are able to finish and retrieve the random key in about 20 minutes. Booking walltime of 1, 2, 3, and 4 hours is too much for M of 15728640 and T of 1024 since the M is total and after being distributed amongst 1023 core it is around 15,375 each.

```
45697767
Lead at t=1 core=1, m=0, key1stB_l_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
```

```
Found CT:_[M-^B][82]__[V][56]__[^\][1c]__[^\][1c]__[.][2e]__[^T][14]_
Recovered Key(EP-1):_[l][6c]__[8][38]__[N][4e]__[y][79]__[P][50]__[E][45]_
Chain SP:_[l][6c]__[8][38]__[N][4e]__[y][79]__[P][50]__[E][45]_
Orig CT:_[^R][12]__[^^][1e]__[2][32]__[d][64]__[M-0][b0]__[‘][60]_
Orig Key:_[l][6c]__[8][38]__[N][4e]__[y][79]__[P][50]__[E][45]_
CT 4 verify:_[^R][12]__[^^][1e]__[2][32]__[d][64]__[M-0][b0]__[‘][60]_
We have Verification also.
Limits: nednodes=2:ppn=8,nodes=2:ppn=8,walltime=04:00:00
Resources:     cput=05:21:43,mem=698032kb,vmem=4267252kb,walltime=00:20:19

45697766
Lead at t=1 core=1, m=0, key1stB_l_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[M-^C][83]__[M-t][f4]__[^T][14]__[Y][59]__[M-^S][93]__[M-+][ab]_
Recovered Key(EP-1):_[l][6c]__[Q][51]__[y][79]__[d][64]__[V][56]__[A][41]_
Chain SP:_[l][6c]__[Q][51]__[y][79]__[d][64]__[V][56]__[A][41]_
Orig CT:_[M-^A][81]__[&][26]__[M-G][c7]__[^L][0c]__[M-o][ef]__[M-!][a1]_
Orig Key:_[l][6c]__[Q][51]__[y][79]__[d][64]__[V][56]__[A][41]_
CT 4 verify:_[M-^A][81]__[&][26]__[M-G][c7]__[^L][0c]__[M-o][ef]__[M-!][a1
   ]_
We have Verification also.
Limits: nednodes=2:ppn=8,nodes=2:ppn=8,walltime=04:00:00
Resources:     cput=05:21:32,mem=696548kb,vmem=4267100kb,walltime=00:20:16

45697765
Lead at t=1 core=1, m=0, key1stB_z_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[M-o][ef]__[ ][20]__[Y][59]__[M-P][d0]__[M-%][a5]__[M-|][fc]_
Recovered Key(EP-1):_[z][7a]__[r][72]__[e][65]__[0][30]__[7][37]__[z][7a]_
Chain SP:_[z][7a]__[r][72]__[e][65]__[0][30]__[7][37]__[z][7a]_
Orig CT:_[M-^Y][99]__[V][56]__[M-^^][9e]__[x][78]__[*][2a]__[;][3b]_
Orig Key:_[z][7a]__[r][72]__[e][65]__[0][30]__[7][37]__[z][7a]_
CT 4 verify:_[M-^Y][99]__[V][56]__[M-^^][9e]__[x][78]__[*][2a]__[;][3b]_
We have Verification also.
Limits: nednodes=2:ppn=8,nodes=2:ppn=8,walltime=04:00:00
Resources:     cput=05:21:47,mem=699244kb,vmem=4267252kb,walltime=00:20:18

45697762
```

```
Lead at t=1 core=1, m=0, key1stB_v_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[Q][51]__[M-^O][8f]__[,][2c]__[{][7b]__[J][4a]__[9][39]_
Recovered Key(EP-1):_[v][76]__[U][55]__[h][68]__[n][6e]__[e][65]__[X][58]_
Chain SP:_[v][76]__[U][55]__[h][68]__[n][6e]__[e][65]__[X][58]_
Orig CT:_[^W][17]__[S][53]__[Y][59]__[M-$][a4]__[M-R][d2]__[,][2c]_
Orig Key:_[v][76]__[U][55]__[h][68]__[n][6e]__[e][65]__[X][58]_
CT 4 verify:_[^W][17]__[S][53]__[Y][59]__[M-$][a4]__[M-R][d2]__[,][2c]_
We have Verification also.
Limits: nednodes=2:ppn=8,nodes=2:ppn=8,walltime=04:00:00
Resources:     cput=05:22:06,mem=699296kb,vmem=4267252kb,walltime=00:20:20

45697764
Lead at t=1 core=1, m=0, key1stB_9_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[M-^Y][99]__[M->][be]__[M-r][f2]__[k][6b]__["][22]__[M-l][ec]_
Recovered Key(EP-1):_[9][39]__[D][44]__[o][6f]__[K][4b]__[q][71]__[S][53]_
Chain SP:_[9][39]__[D][44]__[o][6f]__[K][4b]__[q][71]__[S][53]_
Orig CT:_[E][45]__[M-)][a9]__[M-'][a7]__[M-^W][97]__[M-%][a5]__[#][23]_
Orig Key:_[9][39]__[D][44]__[o][6f]__[K][4b]__[q][71]__[S][53]_
CT 4 verify:_[E][45]__[M-)][a9]__[M-'][a7]__[M-^W][97]__[M-%][a5]__[#][23]
   _
We have Verification also.
Limits: nednodes=2:ppn=8,nodes=2:ppn=8,walltime=04:00:00
Resources:     cput=05:21:40,mem=697504kb,vmem=4267252kb,walltime=00:20:17

45697763
Lead at t=1 core=1, m=0, key1stB_y_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[M-^F][86]__[M-0][b0]__[q][71]__[d][64]__[a][61]__[g][67]_
Recovered Key(EP-1):_[y][79]__[C][43]__[U][55]__[g][67]__[L][4c]__[U][55]_
Chain SP:_[y][79]__[C][43]__[U][55]__[g][67]__[L][4c]__[U][55]_
Orig CT:_[_][5f]__[M-a][e1]__[M-^Z][9a]__[v][76]__[p][70]__[M-[][db]_
Orig Key:_[y][79]__[C][43]__[U][55]__[g][67]__[L][4c]__[U][55]_
CT 4 verify:_[_][5f]__[M-a][e1]__[M-^Z][9a]__[v][76]__[p][70]__[M-[][db]_
We have Verification also.
Limits: nednodes=2:ppn=8,nodes=2:ppn=8,walltime=04:00:00
Resources:     cput=05:21:38,mem=692980kb,vmem=4266176kb,walltime=00:20:16
```

```
45697761
Lead at t=1 core=1, m=0, key1stB_N_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[5][35]__[M][4d]__[5][35]__[M-‘][e0]__[M-}][fd]__[=][3d]_
Recovered Key(EP-1):_[N][4e]__[e][65]__[E][45]__[C][43]__[G][47]__[f][66]_
Chain SP:_[N][4e]__[e][65]__[E][45]__[C][43]__[G][47]__[f][66]_
Orig CT:_[_][5f]__[R][52]__[M-5][b5]__[O][4f]__[M-h][e8]__[M-=][bd]_
Orig Key:_[N][4e]__[e][65]__[E][45]__[C][43]__[G][47]__[f][66]_
CT 4 verify:_[_][5f]__[R][52]__[M-5][b5]__[O][4f]__[M-h][e8]__[M-=][bd]_
We have Verification also.
Limits: nodes=2:ppn=8,nodes=2:ppn=8,walltime=04:00:00
Resources:     cput=05:21:46,mem=696192kb,vmem=4266176kb,walltime=00:20:19

45697759
Lead at t=1 core=1, m=0, key1stB_I_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[M-C][c3]__[5][35]__[M-W][d7]__[1][31]__[1][31]__[M-^_][9f]_
Recovered Key(EP-1):_[I][49]__[X][58]__[A][41]__[q][71]__[i][69]__[N][4e]_
Chain SP:_[I][49]__[X][58]__[A][41]__[q][71]__[i][69]__[N][4e]_
Orig CT:_[M-^Z][9a]__[#][23]__[i][69]__[M-P][d0]__[M-[][db]__[d][64]_
Orig Key:_[I][49]__[X][58]__[A][41]__[q][71]__[i][69]__[N][4e]_
CT 4 verify:_[M-^Z][9a]__[#][23]__[i][69]__[M-P][d0]__[M-[][db]__[d][64]_
We have Verification also.
Limits: nodes=2:ppn=8,nodes=2:ppn=8,walltime=04:00:00
Resources:     cput=05:21:42,mem=695336kb,vmem=4267100kb,walltime=00:20:17

45697760
Lead at t=1 core=1, m=0, key1stB_F_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[]][5d]__[M-#][a3]__[M-j][ea]__[M-y][f9]__[M-^J][8a]__[]][5d]_
Recovered Key(EP-1):_[F][46]__[I][49]__[L][4c]__[8][38]__[s][73]__[2][32]_
Chain SP:_[F][46]__[I][49]__[L][4c]__[8][38]__[s][73]__[2][32]_
Orig CT:_[^G][07]__[M-^L][8c]__[M-o][ef]__[^W][17]__[^E][05]__[M-h][e8]_
Orig Key:_[F][46]__[I][49]__[L][4c]__[8][38]__[s][73]__[2][32]_
CT 4 verify:_[^G][07]__[M-^L][8c]__[M-o][ef]__[^W][17]__[^E][05]__[M-h][e8
   ]_
We have Verification also.
```

```
Limits: neednodes=2:ppn=8,nodes=2:ppn=8,walltime=04:00:00
Resources:     cput=05:21:40,mem=697064kb,vmem=4267100kb,walltime=00:20:16

45697758
Lead at t=1 core=1, m=0, key1stB_I_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[M-w][f7]__[M-K][cb]__[^C][03]__[M-+][ab]__[M-2][b2]__[Q][51]_
Recovered Key(EP-1):_[I][49]__[X][58]__[V][56]__[c][63]__[S][53]__[2][32]_
Chain SP:_[I][49]__[X][58]__[V][56]__[c][63]__[S][53]__[2][32]_
Orig CT:_[2][32]__[-][2d]__[E][45]__[J][4a]__[M-)][a9]__[M-^^][9e]_
Orig Key:_[I][49]__[X][58]__[V][56]__[c][63]__[S][53]__[2][32]_
CT 4 verify:_[2][32]__[-][2d]__[E][45]__[J][4a]__[M-)][a9]__[M-^^][9e]_
We have Verification also.
Limits: neednodes=2:ppn=8,nodes=2:ppn=8,walltime=04:00:00
Resources:     cput=05:21:43,mem=696588kb,vmem=4267100kb,walltime=00:20:17

45697735
Lead at t=1 core=1, m=0, key1stB_k_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[M-|][fc]__[.][2e]__[M-j][ea]__[@][40]__[g][67]__[^E][05]_
Recovered Key(EP-1):_[k][6b]__[6][36]__[S][53]__[r][72]__[P][50]__[S][53]_
Chain SP:_[k][6b]__[6][36]__[S][53]__[r][72]__[P][50]__[S][53]_
Orig CT:_[M-V][d6]__[t][74]__[j][6a]__[M-i][e9]__[x][78]__[M-%][a5]_
Orig Key:_[k][6b]__[6][36]__[S][53]__[r][72]__[P][50]__[S][53]_
CT 4 verify:_[M-V][d6]__[t][74]__[j][6a]__[M-i][e9]__[x][78]__[M-%][a5]_
We have Verification also.
Limits: neednodes=2:ppn=8,nodes=2:ppn=8,walltime=03:00:00
Resources:     cput=05:21:41,mem=699264kb,vmem=4267252kb,walltime=00:20:16

45697737
Lead at t=1 core=1, m=0, key1stB_O_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[^E][05]__[^S][13]__[;][3b]__[V][56]__[^V][16]__[^D][04]_
Recovered Key(EP-1):_[O][4f]__[v][76]__[C][43]__[L][4c]__[7][37]__[q][71]_
Chain SP:_[O][4f]__[v][76]__[C][43]__[L][4c]__[7][37]__[q][71]_
Orig CT:_[M-B][c2]__[M-^D][84]__[M-Y][d9]__[^D][04]__[M-:][ba]__[^S][13]_
Orig Key:_[O][4f]__[v][76]__[C][43]__[L][4c]__[7][37]__[q][71]_
CT 4 verify:_[M-B][c2]__[M-^D][84]__[M-Y][d9]__[^D][04]__[M-:][ba]__[^S
```

```
   ][13]_
We have Verification also.
Limits: neednodes=2:ppn=8,nodes=2:ppn=8,walltime=03:00:00
Resources:     cput=05:21:46,mem=697976kb,vmem=4267252kb,walltime=00:20:18

45697736
Lead at t=1 core=1, m=0, key1stB_H_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[C][43]__[O][4f]__[M-‘][e0]__[M-{][fb]__[M-0][b0]__[^[][1b]_
Recovered Key(EP-1):_[H][48]__[q][71]__[3][33]__[2][32]__[L][4c]__[C][43]_
Chain SP:_[H][48]__[q][71]__[3][33]__[2][32]__[L][4c]__[C][43]_
Orig CT:_[M-m][ed]__[<][3c]__[G][47]__[^U][15]__[M-^A][81]__[X][58]_
Orig Key:_[H][48]__[q][71]__[3][33]__[2][32]__[L][4c]__[C][43]_
CT 4 verify:_[M-m][ed]__[<][3c]__[G][47]__[^U][15]__[M-^A][81]__[X][58]_
We have Verification also.
Limits: neednodes=2:ppn=8,nodes=2:ppn=8,walltime=03:00:00
Resources:     cput=05:21:39,mem=699432kb,vmem=4267252kb,walltime=00:20:18

45697733
Lead at t=1 core=1, m=0, key1stB_X_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[E][45]__[M-=][bd]__[‘][60]__[M-p][f0]__[M-e][e5]__[M-c][e3]_
Recovered Key(EP-1):_[X][58]__[F][46]__[3][33]__[l][6c]__[N][4e]__[1][31]_
Chain SP:_[X][58]__[F][46]__[3][33]__[l][6c]__[N][4e]__[1][31]_
Orig CT:_[M-^O][8f]__[M-<][bc]__[M-.][ae]__[M-X][d8]__[)][29]__[M-(][a8]_
Orig Key:_[X][58]__[F][46]__[3][33]__[l][6c]__[N][4e]__[1][31]_
CT 4 verify:_[M-^O][8f]__[M-<][bc]__[M-.][ae]__[M-X][d8]__[)][29]__[M-(][
   a8]_
We have Verification also.
Limits: neednodes=2:ppn=8,nodes=2:ppn=8,walltime=03:00:00
Resources:     cput=05:21:42,mem=697056kb,vmem=4268276kb,walltime=00:20:19

45697734
Lead at t=1 core=1, m=0, key1stB_B_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[M-^S][93]__[M-i][e9]__[Q][51]__[M-r][f2]__[c][63]__["][22]_
Recovered Key(EP-1):_[B][42]__[u][75]__[M][4d]__[S][53]__[g][67]__[A][41]_
Chain SP:_[B][42]__[u][75]__[M][4d]__[S][53]__[g][67]__[A][41]_
```

```
Orig CT:_[8][38]__[M-^_][9f]__[M-v][f6]__[;][3b]__[M-,][ac]__[P][50]_
Orig Key:_[B][42]__[u][75]__[M][4d]__[S][53]__[g][67]__[A][41]_
CT 4 verify:_[8][38]__[M-^_][9f]__[M-v][f6]__[;][3b]__[M-,][ac]__[P][50]_
We have Verification also.
Limits: nednodes=2:ppn=8,nodes=2:ppn=8,walltime=03:00:00
Resources:     cput=05:21:42,mem=698204kb,vmem=4267252kb,walltime=00:20:19

45697732
Lead at t=1 core=1, m=0, key1stB_5_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[l][6c]__[^T][14]__[k][6b]__[M-c][e3]__[M-4][b4]__[M-o][ef]_
Recovered Key(EP-1):_[5][35]__[j][6a]__[q][71]__[4][34]__[T][54]__[7][37]_
Chain SP:_[5][35]__[j][6a]__[q][71]__[4][34]__[T][54]__[7][37]_
Orig CT:_[7][37]__[B][42]__[1][31]__[M-^Z][9a]__[y][79]__[P][50]_
Orig Key:_[5][35]__[j][6a]__[q][71]__[4][34]__[T][54]__[7][37]_
CT 4 verify:_[7][37]__[B][42]__[1][31]__[M-^Z][9a]__[y][79]__[P][50]_
We have Verification also.
Limits: nednodes=2:ppn=8,nodes=2:ppn=8,walltime=03:00:00
Resources:     cput=05:21:47,mem=694500kb,vmem=4266176kb,walltime=00:20:17

45697731
Lead at t=1 core=1, m=0, key1stB_i_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[M-^H][88]__[R][52]__[M-j][ea]__[^?][7f]__[M-$][a4]__[M-^B][82]_
Recovered Key(EP-1):_[i][69]__[z][7a]__[L][4c]__[f][66]__[0][30]__[P][50]_
Chain SP:_[i][69]__[z][7a]__[L][4c]__[f][66]__[0][30]__[P][50]_
Orig CT:_[M-^A][81]__[^U][15]__[z][7a]__[^?][7f]__[M-^X][98]__[{][7b]_
Orig Key:_[i][69]__[z][7a]__[L][4c]__[f][66]__[0][30]__[P][50]_
CT 4 verify:_[M-^A][81]__[^U][15]__[z][7a]__[^?][7f]__[M-^X][98]__[{][7b]_
We have Verification also.
Limits: nednodes=2:ppn=8,nodes=2:ppn=8,walltime=03:00:00
Resources:     cput=05:21:40,mem=695980kb,vmem=4267252kb,walltime=00:20:15

45697730
Lead at t=1 core=1, m=0, key1stB_h_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[r][72]__[N][4e]__[w][77]__[M-^I][89]__[4][34]__[Z][5a]_
Recovered Key(EP-1):_[h][68]__[V][56]__[g][67]__[F][46]__[L][4c]__[G][47]_
```

```
Chain SP:_[h][68]__[V][56]__[g][67]__[F][46]__[L][4c]__[G][47]_
Orig CT:_[e][65]__[-][2d]__[M][4d]__[H][48]__[S][53]__[)][29]_
Orig Key:_[h][68]__[V][56]__[g][67]__[F][46]__[L][4c]__[G][47]_
CT 4 verify:_[e][65]__[-][2d]__[M][4d]__[H][48]__[S][53]__[)][29]_
We have Verification also.
Limits: nednodes=2:ppn=8,nodes=2:ppn=8,walltime=03:00:00
Resources:     cput=05:21:46,mem=697908kb,vmem=4267252kb,walltime=00:20:18

45697728
Lead at t=1 core=1, m=0, key1stB_L_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[M-S][d3]__[M-|][fc]__[M-1][b1]__[^G][07]__[K][4b]__[o][6f]_
Recovered Key(EP-1):_[L][4c]__[p][70]__[w][77]__[3][33]__[y][79]__[P][50]_
Chain SP:_[L][4c]__[p][70]__[w][77]__[3][33]__[y][79]__[P][50]_
Orig CT:_[m][6d]__[{][7b]__[M-4][b4]__[M-,][ac]__[Z][5a]__[M-]][dd]_
Orig Key:_[L][4c]__[p][70]__[w][77]__[3][33]__[y][79]__[P][50]_
CT 4 verify:_[m][6d]__[{][7b]__[M-4][b4]__[M-,][ac]__[Z][5a]__[M-]][dd]_
We have Verification also.
Limits: nednodes=2:ppn=8,nodes=2:ppn=8,walltime=03:00:00
Resources:     cput=05:21:40,mem=696836kb,vmem=4266572kb,walltime=00:20:16

45697729
Lead at t=1 core=1, m=0, key1stB_g_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[M-u][f5]__[M-=][bd]__[s][73]__[‘][60]__[M-v][f6]__[M-f][e6]_
Recovered Key(EP-1):_[g][67]__[U][55]__[q][71]__[2][32]__[6][36]__[F][46]_
Chain SP:_[g][67]__[U][55]__[q][71]__[2][32]__[6][36]__[F][46]_
Orig CT:_[M-X][d8]__[X][58]__[M-d][e4]__[M-H][c8]__[M-$][a4]__[^Y][19]_
Orig Key:_[g][67]__[U][55]__[q][71]__[2][32]__[6][36]__[F][46]_
CT 4 verify:_[M-X][d8]__[X][58]__[M-d][e4]__[M-H][c8]__[M-$][a4]__[^Y][19]
   _
We have Verification also.
Limits: nednodes=2:ppn=8,nodes=2:ppn=8,walltime=03:00:00
Resources:     cput=05:21:18,mem=697040kb,vmem=4267100kb,walltime=00:20:14

45697706
Lead at t=1 core=1, m=0, key1stB_R_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
```

```
Found CT:_[M-]][dd]__[+][2b]__[M-;][bb]__[M-^C][83]__[M-^V][96]__[c][63]_
Recovered Key(EP-1):_[R][52]__[9][39]__[f][66]__[A][41]__[W][57]__[1][31]_
Chain SP:_[R][52]__[9][39]__[f][66]__[A][41]__[W][57]__[1][31]_
Orig CT:_[*][2a]__[M-‘][e0]__[M-^V][96]__[M-g][e7]__[!][21]__[M-c][e3]_
Orig Key:_[R][52]__[9][39]__[f][66]__[A][41]__[W][57]__[1][31]_
CT 4 verify:_[*][2a]__[M-‘][e0]__[M-^V][96]__[M-g][e7]__[!][21]__[M-c][e3]
   _
We have Verification also.
Limits: nednodes=2:ppn=8,nodes=2:ppn=8,walltime=02:00:00
Resources:     cput=05:21:43,mem=699524kb,vmem=4268276kb,walltime=00:20:18

45697707
Lead at t=1 core=1, m=0, key1stB_m_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[M->][be]__[}][7d]__["][22]__[^C][03]__[M-~][fe]__[M-\][dc]_
Recovered Key(EP-1):_[m][6d]__[Y][59]__[c][63]__[G][47]__[g][67]__[J][4a]_
Chain SP:_[m][6d]__[Y][59]__[c][63]__[G][47]__[g][67]__[J][4a]_
Orig CT:_[T][54]__[M-h][e8]__[W][57]__[M-\][dc]__[M-l][ec]__[M-^Q][91]_
Orig Key:_[m][6d]__[Y][59]__[c][63]__[G][47]__[g][67]__[J][4a]_
CT 4 verify:_[T][54]__[M-h][e8]__[W][57]__[M-\][dc]__[M-l][ec]__[M-^Q][91]
   _
We have Verification also.
Limits: nednodes=2:ppn=8,nodes=2:ppn=8,walltime=02:00:00
Resources:     cput=05:21:19,mem=697228kb,vmem=4267252kb,walltime=00:20:15

45697704
Lead at t=1 core=1, m=0, key1stB_L_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[M->][be]__[?][3f]__[M-H][c8]__[5][35]__[M-+][ab]__[g][67]_
Recovered Key(EP-1):_[L][4c]__[B][42]__[K][4b]__[W][57]__[5][35]__[e][65]_
Chain SP:_[L][4c]__[B][42]__[K][4b]__[W][57]__[5][35]__[e][65]_
Orig CT:_[M-d][e4]__[M-i][e9]__[M-e][e5]__[M-%][a5]__[T][54]__[M-S][d3]_
Orig Key:_[L][4c]__[B][42]__[K][4b]__[W][57]__[5][35]__[e][65]_
CT 4 verify:_[M-d][e4]__[M-i][e9]__[M-e][e5]__[M-%][a5]__[T][54]__[M-S][d3
   ]_
We have Verification also.
Limits: nednodes=2:ppn=8,nodes=2:ppn=8,walltime=02:00:00
Resources:     cput=05:21:45,mem=696512kb,vmem=4267100kb,walltime=00:20:19
```

```
45697705
Lead at t=1 core=1, m=0, key1stB_A_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[M-/][af]__[M-Z][da]__[^?][7f]__[M-T][d4]__[M-I][c9]__[M-9][b9]_
Recovered Key(EP-1):_[A][41]__[R][52]__[O][4f]__[q][71]__[s][73]__[l][6c]_
Chain SP:_[A][41]__[R][52]__[O][4f]__[q][71]__[s][73]__[l][6c]_
Orig CT:_[^G][07]__[M-x][f8]__[3][33]__[k][6b]__[^O][0f]__[M-1][b1]_
Orig Key:_[A][41]__[R][52]__[O][4f]__[q][71]__[s][73]__[l][6c]_
CT 4 verify:_[^G][07]__[M-x][f8]__[3][33]__[k][6b]__[^O][0f]__[M-1][b1]_
We have Verification also.
Limits: nednodes=2:ppn=8,nodes=2:ppn=8,walltime=02:00:00
Resources:     cput=05:21:41,mem=698224kb,vmem=4267252kb,walltime=00:20:17

45697703
Lead at t=1 core=1, m=0, key1stB_2_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[%][25]__[M-(][a8]__[M-^N][8e]__[M-m][ed]__[,][2c]__[a][61]_
Recovered Key(EP-1):_[2][32]__[u][75]__[2][32]__[z][7a]__[f][66]__[L][4c]_
Chain SP:_[2][32]__[u][75]__[2][32]__[z][7a]__[f][66]__[L][4c]_
Orig CT:_[w][77]__[C][43]__[^P][10]__[u][75]__[M-^V][96]__[a][61]_
Orig Key:_[2][32]__[u][75]__[2][32]__[z][7a]__[f][66]__[L][4c]_
CT 4 verify:_[w][77]__[C][43]__[^P][10]__[u][75]__[M-^V][96]__[a][61]_
We have Verification also.
Limits: nednodes=2:ppn=8,nodes=2:ppn=8,walltime=02:00:00
Resources:     cput=05:21:40,mem=693192kb,vmem=4267100kb,walltime=00:20:20

45697702
Lead at t=1 core=1, m=0, key1stB_I_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[M-T][d4]__[5][35]__[M-c][e3]__[M-P][d0]__[2][32]__[7][37]_
Recovered Key(EP-1):_[I][49]__[V][56]__[h][68]__[4][34]__[9][39]__[8][38]_
Chain SP:_[I][49]__[V][56]__[h][68]__[4][34]__[9][39]__[8][38]_
Orig CT:_[C][43]__[M-^?][ff]__[M->][be]__[M-^N][8e]__[b][62]__[M-^P][90]_
Orig Key:_[I][49]__[V][56]__[h][68]__[4][34]__[9][39]__[8][38]_
CT 4 verify:_[C][43]__[M-^?][ff]__[M->][be]__[M-^N][8e]__[b][62]__[M-^P
   ][90]_
We have Verification also.
```

```
Limits: neednodes=2:ppn=8,nodes=2:ppn=8,walltime=02:00:00
Resources:     cput=05:22:00,mem=697940kb,vmem=4267100kb,walltime=00:20:18

45697701
Lead at t=1 core=1, m=0, key1stB_E_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[o][6f]__[M-^W][97]__[^_][1f]__[M-/][af]__[M-J][ca]__[M-I][c9]_
Recovered Key(EP-1):_[E][45]__[0][30]__[a][61]__[w][77]__[y][79]__[5][35]_
Chain SP:_[E][45]__[0][30]__[a][61]__[w][77]__[y][79]__[5][35]_
Orig CT:_[M-%][a5]__[M-1][b1]__[M-^Q][91]__[M-_][df]__[Z][5a]__[M--][ad]_
Orig Key:_[E][45]__[0][30]__[a][61]__[w][77]__[y][79]__[5][35]_
CT 4 verify:_[M-%][a5]__[M-1][b1]__[M-^Q][91]__[M-_][df]__[Z][5a]__[M--][
   ad]_
We have Verification also.
Limits: neednodes=2:ppn=8,nodes=2:ppn=8,walltime=02:00:00
Resources:     cput=05:21:55,mem=698212kb,vmem=4267252kb,walltime=00:20:20

45697700
Lead at t=1 core=1, m=0, key1stB_m_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[4][34]__[M-}][fd]__[M-N][ce]__[M-^M][8d]__[M-^D][84]__[B][42]_
Recovered Key(EP-1):_[m][6d]__[9][39]__[s][73]__[h][68]__[T][54]__[c][63]_
Chain SP:_[m][6d]__[9][39]__[s][73]__[h][68]__[T][54]__[c][63]_
Orig CT:_[M- ][a0]__[M-^D][84]__[$][24]__[M-?][bf]__[M-1][b1]__[M-^^][9e]_
Orig Key:_[m][6d]__[9][39]__[s][73]__[h][68]__[T][54]__[c][63]_
CT 4 verify:_[M- ][a0]__[M-^D][84]__[$][24]__[M-?][bf]__[M-1][b1]__[M
   -^^][9e]_
We have Verification also.
Limits: neednodes=2:ppn=8,nodes=2:ppn=8,walltime=02:00:00
Resources:     cput=05:21:40,mem=696752kb,vmem=4267100kb,walltime=00:20:17

45697699
Lead at t=1 core=1, m=0, key1stB_J_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[?][3f]__[M-&][a6]__[M-^Q][91]__[^@][00]__[M-x][f8]__[i][69]_
Recovered Key(EP-1):_[J][4a]__[R][52]__[9][39]__[j][6a]__[5][35]__[T][54]_
Chain SP:_[J][4a]__[R][52]__[9][39]__[j][6a]__[5][35]__[T][54]_
Orig CT:_[M--][ad]__[M--][ad]__[M][4d]__[]][5d]__[^T][14]__[.][2e]_
```

```
Orig Key:_[J][4a]__[R][52]__[9][39]__[j][6a]__[5][35]__[T][54]_
CT 4 verify:_[M--][ad]__[M--][ad]__[M][4d]__[]][5d]__[^T][14]__[.][2e]_
We have Verification also.
Limits: nodes=2:ppn=8,nodes=2:ppn=8,walltime=02:00:00
Resources:     cput=05:21:41,mem=693484kb,vmem=4266176kb,walltime=00:20:18

45697698
Lead at t=1 core=1, m=0, key1stB_o_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[h][68]__[M- ][a0]__[M-k][eb]__[N][4e]__[o][6f]__[M-^\][9c]_
Recovered Key(EP-1):_[o][6f]__[S][53]__[P][50]__[C][43]__[9][39]__[u][75]_
Chain SP:_[o][6f]__[S][53]__[P][50]__[C][43]__[9][39]__[u][75]_
Orig CT:_[\][5c]__[c][63]__[M-^M][8d]__[M-l][ec]__[M-I][c9]__[M-7][b7]_
Orig Key:_[o][6f]__[S][53]__[P][50]__[C][43]__[9][39]__[u][75]_
CT 4 verify:_[\][5c]__[c][63]__[M-^M][8d]__[M-l][ec]__[M-I][c9]__[M-7][b7]
   _
We have Verification also.
Limits: nodes=2:ppn=8,nodes=2:ppn=8,walltime=02:00:00
Resources:     cput=05:21:22,mem=694936kb,vmem=4266176kb,walltime=00:20:15

45697676
Lead at t=1 core=1, m=0, key1stB_h_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[]][5d]__[M-/][af]__[M-^U][95]__[’][27]__[>][3e]__[!][21]_
Recovered Key(EP-1):_[h][68]__[S][53]__[x][78]__[C][43]__[V][56]__[Z][5a]_
Chain SP:_[h][68]__[S][53]__[x][78]__[C][43]__[V][56]__[Z][5a]_
Orig CT:_[5][35]__["][22]__[5][35]__[&][26]__[)][29]__[^Z][1a]_
Orig Key:_[h][68]__[S][53]__[x][78]__[C][43]__[V][56]__[Z][5a]_
CT 4 verify:_[5][35]__["][22]__[5][35]__[&][26]__[)][29]__[^Z][1a]_
We have Verification also.
Limits: nodes=2:ppn=8,nodes=2:ppn=8,walltime=01:00:00
Resources:     cput=05:21:39,mem=698340kb,vmem=4267100kb,walltime=00:20:18

45697677
Lead at t=1 core=1, m=0, key1stB_T_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[M-^C][83]__[’][27]__[M-^F][86]__[M-^M][8d]__[M-.][ae]__[V][56]_
Recovered Key(EP-1):_[T][54]__[X][58]__[U][55]__[N][4e]__[c][63]__[0][30]_
```

```
Chain SP:_[T][54]__[X][58]__[U][55]__[N][4e]__[c][63]__[0][30]_
Orig CT:_[t][74]__[M-.][ae]__[M- ][a0]__[M-^Z][9a]__[M-‘][e0]__[M-^B][82]_
Orig Key:_[T][54]__[X][58]__[U][55]__[N][4e]__[c][63]__[0][30]_
CT 4 verify:_[t][74]__[M-.][ae]__[M- ][a0]__[M-^Z][9a]__[M-‘][e0]__[M-^B
   ][82]_
We have Verification also.
Limits: nednodes=2:ppn=8,nodes=2:ppn=8,walltime=01:00:00
Resources:     cput=05:21:24,mem=697360kb,vmem=4268052kb,walltime=00:20:14

45697672
Lead at t=1 core=1, m=0, key1stB_n_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[M-3][b3]__[G][47]__[^P][10]__[M-I][c9]__[z][7a]__[M-x][f8]_
Recovered Key(EP-1):_[n][6e]__[O][4f]__[k][6b]__[I][49]__[R][52]__[v][76]_
Chain SP:_[n][6e]__[O][4f]__[k][6b]__[I][49]__[R][52]__[v][76]_
Orig CT:_[M-E][c5]__[F][46]__[B][42]__[\][5c]__[M-^C][83]__[^F][06]_
Orig Key:_[n][6e]__[O][4f]__[k][6b]__[I][49]__[R][52]__[v][76]_
CT 4 verify:_[M-E][c5]__[F][46]__[B][42]__[\][5c]__[M-^C][83]__[^F][06]_
We have Verification also.
Limits: nednodes=2:ppn=8,nodes=2:ppn=8,walltime=01:00:00
Resources:     cput=05:21:43,mem=693680kb,vmem=4266176kb,walltime=00:20:17

45697673
Lead at t=1 core=1, m=0, key1stB_L_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[M-^S][93]__[M-^@][80]__[M-^C][83]__[M-^\][9c]__[a][61]__[M-Q][
   d1]_
Recovered Key(EP-1):_[L][4c]__[Z][5a]__[d][64]__[s][73]__[Q][51]__[u][75]_
Chain SP:_[L][4c]__[Z][5a]__[d][64]__[s][73]__[Q][51]__[u][75]_
Orig CT:_["][22]__[^_][1f]__[r][72]__[M-q][f1]__[^O][0f]__[}][7d]_
Orig Key:_[L][4c]__[Z][5a]__[d][64]__[s][73]__[Q][51]__[u][75]_
CT 4 verify:_["][22]__[^_][1f]__[r][72]__[M-q][f1]__[^O][0f]__[}][7d]_
We have Verification also.
Limits: nednodes=2:ppn=8,nodes=2:ppn=8,walltime=01:00:00
Resources:     cput=05:21:42,mem=697152kb,vmem=4266976kb,walltime=00:20:17

45697671
Lead at t=1 core=1, m=0, key1stB_6_
```

```
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[m][6d]__[M-^A][81]__[^Z][1a]__[M-f][e6]__[M->][be]__[.][2e]_
Recovered Key(EP-1):_[6][36]__[c][63]__[d][64]__[l][6c]__[K][4b]__[R][52]_
Chain SP:_[6][36]__[c][63]__[d][64]__[l][6c]__[K][4b]__[R][52]_
Orig CT:_[^W][17]__[T][54]__[M-l][ec]__[M-^U][95]__[^D][04]__[M-a][e1]_
Orig Key:_[6][36]__[c][63]__[d][64]__[l][6c]__[K][4b]__[R][52]_
CT 4 verify:_[^W][17]__[T][54]__[M-l][ec]__[M-^U][95]__[^D][04]__[M-a][e1]
   _
We have Verification also.
Limits: nednodes=2:ppn=8,nodes=2:ppn=8,walltime=01:00:00
Resources:     cput=05:21:16,mem=698636kb,vmem=4267252kb,walltime=00:20:15

45697674
Lead at t=1 core=1, m=0, key1stB_a_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[M-/][af]__[M- ][a0]__[M-k][eb]__[M-‘][e0]__[M-~][fe]__[^S][13]_
Recovered Key(EP-1):_[a][61]__[u][75]__[6][36]__[o][6f]__[y][79]__[C][43]_
Chain SP:_[a][61]__[u][75]__[6][36]__[o][6f]__[y][79]__[C][43]_
Orig CT:_[0][30]__[M-^I][89]__[^C][03]__[n][6e]__[M-x][f8]__[M-3][b3]_
Orig Key:_[a][61]__[u][75]__[6][36]__[o][6f]__[y][79]__[C][43]_
CT 4 verify:_[0][30]__[M-^I][89]__[^C][03]__[n][6e]__[M-x][f8]__[M-3][b3]_
We have Verification also.
Limits: nednodes=2:ppn=8,nodes=2:ppn=8,walltime=01:00:00
Resources:     cput=05:21:40,mem=695268kb,vmem=4267100kb,walltime=00:20:16

45697675
Lead at t=1 core=1, m=0, key1stB_W_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[C][43]__[M-^H][88]__[M-f][e6]__[X][58]__[T][54]__[d][64]_
Recovered Key(EP-1):_[W][57]__[e][65]__[I][49]__[G][47]__[7][37]__[w][77]_
Chain SP:_[W][57]__[e][65]__[I][49]__[G][47]__[7][37]__[w][77]_
Orig CT:_[^E][05]__[M-^W][97]__["][22]__[4][34]__[M-$][a4]__[^R][12]_
Orig Key:_[W][57]__[e][65]__[I][49]__[G][47]__[7][37]__[w][77]_
CT 4 verify:_[^E][05]__[M-^W][97]__["][22]__[4][34]__[M-$][a4]__[^R][12]_
We have Verification also.
Limits: nednodes=2:ppn=8,nodes=2:ppn=8,walltime=01:00:00
Resources:     cput=05:21:23,mem=697892kb,vmem=4267252kb,walltime=00:20:15
```

```
45697668
Lead at t=1 core=1, m=0, key1stB_4_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[N][4e]__[^O][0f]__[M-"][a2]__[^C][03]__[M-G][c7]__[M-i][e9]_
Recovered Key(EP-1):_[4][34]__[U][55]__[8][38]__[5][35]__[T][54]__[p][70]_
Chain SP:_[4][34]__[U][55]__[8][38]__[5][35]__[T][54]__[p][70]_
Orig CT:_[%][25]__[e][65]__[M-c][e3]__()][29]__[^C][03]__[M-W][d7]_
Orig Key:_[4][34]__[U][55]__[8][38]__[5][35]__[T][54]__[p][70]_
CT 4 verify:_[%][25]__[e][65]__[M-c][e3]__()][29]__[^C][03]__[M-W][d7]_
We have Verification also.
Limits: nednodes=2:ppn=8,nodes=2:ppn=8,walltime=01:00:00
Resources:     cput=05:21:45,mem=698808kb,vmem=4267260kb,walltime=00:20:18

45697670
Lead at t=1 core=1, m=0, key1stB_B_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[k][6b]__[0][30]__[l][6c]__[M-Q][d1]__[M-J][ca]__[W][57]_
Recovered Key(EP-1):_[B][42]__[t][74]__[N][4e]__[s][73]__[V][56]__[y][79]_
Chain SP:_[B][42]__[t][74]__[N][4e]__[s][73]__[V][56]__[y][79]_
Orig CT:_[M-P][d0]__[M-5][b5]__[M-)][a9]__[M-^H][88]__[M-5][b5]__[^_][1f]_
Orig Key:_[B][42]__[t][74]__[N][4e]__[s][73]__[V][56]__[y][79]_
CT 4 verify:_[M-P][d0]__[M-5][b5]__[M-)][a9]__[M-^H][88]__[M-5][b5]__[^_
   ][1f]_
We have Verification also.
Limits: nednodes=2:ppn=8,nodes=2:ppn=8,walltime=01:00:00
Resources:     cput=05:21:40,mem=698888kb,vmem=4267252kb,walltime=00:20:17

45697669
Lead at t=1 core=1, m=0, key1stB_e_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[M-^X][98]__[A][41]__[M-#][a3]__[W][57]__[M-{][fb]__[M-E][c5]_
Recovered Key(EP-1):_[e][65]__[K][4b]__[V][56]__[A][41]__[z][7a]__[b][62]_
Chain SP:_[e][65]__[K][4b]__[V][56]__[A][41]__[z][7a]__[b][62]_
Orig CT:_[^A][01]__[M-#][a3]__[B][42]__[k][6b]__[I][49]__[y][79]_
Orig Key:_[e][65]__[K][4b]__[V][56]__[A][41]__[z][7a]__[b][62]_
CT 4 verify:_[^A][01]__[M-#][a3]__[B][42]__[k][6b]__[I][49]__[y][79]_
We have Verification also.
Limits: nednodes=2:ppn=8,nodes=2:ppn=8,walltime=01:00:00
```

```
Resources:      cput=05:21:36,mem=697880kb,vmem=4267100kb,walltime=00:20:15
```

A.1.5 The 2G M batch

Job ID 45697538

```
The number of booked cores is 1024 (1 Parent and 1023 children)
The job was finished in 0:35:29 minutes but did not find the Key.
plaintext is: PTPTPT
Random key: [0_53_S][1_39_9][2_46_F][3_4e_N][4_71_q][5_72_r] and it is
   used as the start point of the first chain.
After encrypting the plaintext T times (1024) the chain end point becomes
   : ___[0c]___[12]___[9b]___[41]___[f0]___[6f]
Ciphertext Challenge: [0_cc_][1_83_~C][2 _e1_ ][3_14_^T][4_4f_O][5_8d_~M]
CT decrypts back to: [0_50_P][1_54_T][2_50_P][3_54_T][4_50_P][5_54_T]
output log: tmto1024t1024c2145386496m2hrs_bestCvrg0.o45697538
Output size (printf overhead): 43101 lines, 3321620 characters.
error log: tmto1024t1024c2145386496m2hrs_bestCvrg0.e45697538 and it was
   empty.
Job log: tmt_Fri_Oct_20_21_18_33_EDT_2017_M=2145386496_C=1024_W=2_00_00_T
   =1024_B=48_E=2_H=0_L=0_S=1_R=GPC__notes=
   tmto1024t1024c2145386496m2hrs_bestCvrg0_45697538.gpc-sched-ib0.joblog
source code file: tmt_Fri_Oct_20_21_18_33_EDT_2017_M=2145386496_C=1024_W=2
   _00_00_T=1024_B=48_E=2_H=0_L=0_S=1_R=GPC__notes=
   tmto1024t1024c2145386496m2hrs_bestCvrg0.c
job submission script: tmt_Fri_Oct_20_21_18_33_EDT_2017_M=2145386496_C
   =1024_W=2_00_00_T=1024_B=48_E=2_H=0_L=0_S=1_R=GPC__notes=
   tmto1024t1024c2145386496m2hrs_bestCvrg0.sh
object file: tmt_Fri_Oct_20_21_18_33_EDT_2017_M=2145386496_C=1024_W=2
   _00_00_T=1024_B=48_E=2_H=0_L=0_S=1_R=GPC__notes=
   tmto1024t1024c2145386496m2hrs_bestCvrg0.out
Cores: 1024
usableCores: 1023
PID:589
Date: tmt_Fri_Oct_20_21_18_33_EDT_2017
M=2145386496
```

```
C=1024
W=2:00:00
T=1024
B=48
E=2
H=0
L=0
S=1
R=GPC
notes=tmto1024t1024c2145386496m2hrs_bestCvrg0.out
```

Observations: There was and issue with the verification code. Jobs o45697537, o45697536, o45697535, o45697534, o45697533, o45697532, o45697531, o45697530 act similarly where the search is done in 35 minutes but now results were found.

```
o45697530:Limits:     neednodes=128:ppn=8,nodes=128:ppn=8,walltime=02:00:00
o45697530:Resources:  cput=598:52:01,mem=236339120kb,vmem=423044068kb,
   walltime=00:35:25
o45697531:Limits:     neednodes=128:ppn=8,nodes=128:ppn=8,walltime=02:00:00
o45697531:Resources:  cput=598:55:12,mem=236322336kb,vmem=423044144kb,
   walltime=00:35:23
o45697532:Limits:     neednodes=128:ppn=8,nodes=128:ppn=8,walltime=02:00:00
o45697532:Resources:  cput=598:52:45,mem=236442912kb,vmem=423044152kb,
   walltime=00:35:25
o45697533:Limits:     neednodes=128:ppn=8,nodes=128:ppn=8,walltime=02:00:00
o45697533:Resources:  cput=598:55:38,mem=236337012kb,vmem=423044092kb,
   walltime=00:35:24
o45697534:Limits:     neednodes=128:ppn=8,nodes=128:ppn=8,walltime=02:00:00
o45697534:Resources:  cput=598:53:03,mem=236650348kb,vmem=423044116kb,
   walltime=00:35:26
o45697535:Limits:     neednodes=128:ppn=8,nodes=128:ppn=8,walltime=02:00:00
o45697535:Resources:  cput=598:54:32,mem=236374016kb,vmem=423043960kb,
   walltime=00:35:26
o45697536:Limits:     neednodes=128:ppn=8,nodes=128:ppn=8,walltime=02:00:00
o45697536:Resources:  cput=598:51:31,mem=236685612kb,vmem=423044084kb,
   walltime=00:35:29
o45697537:Limits:     neednodes=128:ppn=8,nodes=128:ppn=8,walltime=02:00:00
o45697537:Resources:  cput=598:52:11,mem=236349688kb,vmem=423044104kb,
```

```
   walltime=00:35:26
o45697538:Limits:     neednodes=128:ppn=8,nodes=128:ppn=8,walltime=02:00:00
o45697538:Resources:  cput=599:35:08,mem=236362516kb,vmem=423044100kb,
   walltime=00:35:29
```

No leads or partial matches were found which enforces the conclusion that it was a bug with setting up the random key. Following the (walltime range) clustering method we can assume the same behavior will happen in the followin jobs (45787000 45786997 45786996 45786995 45786994 45786993 45786992 45786991 45697620 45697619 45697618 45697617 45697616 45697615 45697614 45697613 45697612 45697611 45697580 45697579 45697578 45697577 45697576 45697575 45697574 45697573 45697571 45697572 45697499 45697498 45697497 45697496 45697494 45697493 45697492 45697491 45697490 45697489)

```
45787000
Limits: neednodes=128:ppn=8,nodes=128:ppn=8,walltime=04:00:00
Resources:     cput=598:46:37,mem=236405260kb,vmem=423044148kb,walltime
   =00:35:25

45786997
Limits: neednodes=128:ppn=8,nodes=128:ppn=8,walltime=04:00:00
Resources:     cput=598:53:04,mem=236480528kb,vmem=423044064kb,walltime
   =00:35:22

45786996
Limits: neednodes=128:ppn=8,nodes=128:ppn=8,walltime=04:00:00
Resources:     cput=598:51:56,mem=236627076kb,vmem=423044056kb,walltime
   =00:35:24

45786995
Limits: neednodes=128:ppn=8,nodes=128:ppn=8,walltime=04:00:00
Resources:     cput=598:47:10,mem=236382676kb,vmem=423043952kb,walltime
   =00:35:33

45786994
Limits: neednodes=128:ppn=8,nodes=128:ppn=8,walltime=04:00:00
Resources:     cput=598:50:41,mem=236276364kb,vmem=423044136kb,walltime
   =00:35:33
```

```
45786993
Limits: neednodes=128:ppn=8,nodes=128:ppn=8,walltime=04:00:00
Resources:     cput=598:46:23,mem=236348016kb,vmem=423044128kb,walltime
   =00:35:21

45786992
Limits: neednodes=128:ppn=8,nodes=128:ppn=8,walltime=04:00:00
Resources:     cput=599:04:12,mem=236305548kb,vmem=423044004kb,walltime
   =00:35:28

45786991
Limits: neednodes=128:ppn=8,nodes=128:ppn=8,walltime=04:00:00
Resources:     cput=598:50:46,mem=236421868kb,vmem=423043916kb,walltime
   =00:35:36

45697620
Limits: neednodes=128:ppn=8,nodes=128:ppn=8,walltime=04:00:00
Resources:     cput=598:43:03,mem=236291092kb,vmem=423044096kb,walltime
   =00:35:20

45697619
Limits: neednodes=128:ppn=8,nodes=128:ppn=8,walltime=04:00:00
Resources:     cput=598:42:16,mem=236553812kb,vmem=423044240kb,walltime
   =00:35:22

45697618
Limits: neednodes=128:ppn=8,nodes=128:ppn=8,walltime=04:00:00
Resources:     cput=598:44:13,mem=236241296kb,vmem=423044140kb,walltime
   =00:35:19

45697617
Limits: neednodes=128:ppn=8,nodes=128:ppn=8,walltime=04:00:00
Resources:     cput=598:39:34,mem=236344692kb,vmem=423043976kb,walltime
   =00:35:22

45697616
Limits: neednodes=128:ppn=8,nodes=128:ppn=8,walltime=04:00:00
Resources:     cput=598:39:15,mem=236379556kb,vmem=423044160kb,walltime
```

```
   =00:35:19

45697615
Limits: neednodes=128:ppn=8,nodes=128:ppn=8,walltime=04:00:00
Resources:     cput=598:39:13,mem=236250988kb,vmem=423044144kb,walltime
   =00:35:18

45697614
Limits: neednodes=128:ppn=8,nodes=128:ppn=8,walltime=04:00:00
Resources:     cput=598:39:49,mem=236289704kb,vmem=423044092kb,walltime
   =00:35:20

45697613
Limits: neednodes=128:ppn=8,nodes=128:ppn=8,walltime=04:00:00
Resources:     cput=598:37:39,mem=236319892kb,vmem=423044088kb,walltime
   =00:35:20

45697612
Limits: neednodes=128:ppn=8,nodes=128:ppn=8,walltime=04:00:00
Resources:     cput=598:40:13,mem=236294420kb,vmem=423044124kb,walltime
   =00:35:20

45697611
Limits: neednodes=128:ppn=8,nodes=128:ppn=8,walltime=04:00:00
Resources:     cput=599:01:18,mem=236315880kb,vmem=423044112kb,walltime
   =00:35:20

45697580
Limits: neednodes=128:ppn=8,nodes=128:ppn=8,walltime=03:00:00
Resources:     cput=598:41:10,mem=236493236kb,vmem=423044208kb,walltime
   =00:35:20

45697579
Limits: neednodes=128:ppn=8,nodes=128:ppn=8,walltime=03:00:00
Resources:     cput=598:44:52,mem=236321268kb,vmem=423044108kb,walltime
   =00:35:20

45697578
```

```
Limits: neednodes=128:ppn=8,nodes=128:ppn=8,walltime=03:00:00
Resources:     cput=599:12:31,mem=236292240kb,vmem=423044092kb,walltime
   =00:35:23

45697577
Limits: neednodes=128:ppn=8,nodes=128:ppn=8,walltime=03:00:00
Resources:     cput=598:44:14,mem=236212560kb,vmem=423044052kb,walltime
   =00:35:20

45697576
Limits: neednodes=128:ppn=8,nodes=128:ppn=8,walltime=03:00:00
Resources:     cput=598:42:42,mem=236289500kb,vmem=423044144kb,walltime
   =00:35:22

45697575
Limits: neednodes=128:ppn=8,nodes=128:ppn=8,walltime=03:00:00
Resources:     cput=598:45:42,mem=236437144kb,vmem=423044120kb,walltime
   =00:35:23

45697574
Limits: neednodes=128:ppn=8,nodes=128:ppn=8,walltime=03:00:00
Resources:     cput=599:06:57,mem=236236796kb,vmem=423044116kb,walltime
   =00:35:25

45697573
Limits: neednodes=128:ppn=8,nodes=128:ppn=8,walltime=03:00:00
Resources:     cput=598:36:39,mem=236316844kb,vmem=423044100kb,walltime
   =00:35:22

45697571
Limits: neednodes=128:ppn=8,nodes=128:ppn=8,walltime=03:00:00
Resources:     cput=599:10:21,mem=236351020kb,vmem=423044116kb,walltime
   =00:35:26

45697572
Limits: neednodes=128:ppn=8,nodes=128:ppn=8,walltime=03:00:00
Resources:     cput=598:43:52,mem=236365600kb,vmem=423044088kb,walltime
   =00:35:25
```

```
45697499
Limits: nednodes=128:ppn=8,nodes=128:ppn=8,walltime=01:00:00
Resources:     cput=598:51:16,mem=236333596kb,vmem=423044052kb,walltime
   =00:35:26

45697498
Limits: nednodes=128:ppn=8,nodes=128:ppn=8,walltime=01:00:00
Resources:     cput=598:50:58,mem=236447952kb,vmem=423044084kb,walltime
   =00:35:28

45697497
Limits: nednodes=128:ppn=8,nodes=128:ppn=8,walltime=01:00:00
Resources:     cput=598:53:47,mem=236449800kb,vmem=423044156kb,walltime
   =00:35:22

45697496
Limits: nednodes=128:ppn=8,nodes=128:ppn=8,walltime=01:00:00
Resources:     cput=598:49:23,mem=236487644kb,vmem=423043968kb,walltime
   =00:35:28

45697494
Limits: nednodes=128:ppn=8,nodes=128:ppn=8,walltime=01:00:00
Resources:     cput=598:53:50,mem=236328496kb,vmem=423044112kb,walltime
   =00:35:23

45697493
Limits: nednodes=128:ppn=8,nodes=128:ppn=8,walltime=01:00:00
Resources:     cput=598:55:38,mem=236534824kb,vmem=423043992kb,walltime
   =00:35:25

45697492
Limits: nednodes=128:ppn=8,nodes=128:ppn=8,walltime=01:00:00
Resources:     cput=598:56:49,mem=236355932kb,vmem=423044144kb,walltime
   =00:35:24

45697491
Limits: nednodes=128:ppn=8,nodes=128:ppn=8,walltime=01:00:00
```

```
Resources:     cput=598:54:20,mem=236536236kb,vmem=423043964kb,walltime
   =00:35:24

45697490
Limits: neednodes=128:ppn=8,nodes=128:ppn=8,walltime=01:00:00
Resources:     cput=598:51:39,mem=236404932kb,vmem=423044080kb,walltime
   =00:35:28

45697489
Limits: neednodes=128:ppn=8,nodes=128:ppn=8,walltime=01:00:00
Resources:     cput=598:52:50,mem=236450772kb,vmem=423044208kb,walltime
   =00:35:26
```

Observation: The all finish without being interrupted by GPC in 35 minutes without finding the key.

A.1.6 Million M per core 1024t 8c 7,340,032m

Job ID 45697756 45697757 45697755 45697754 45697753 45697752 45697751 45697750 45697749 45697748 45697727 45697726 45697724 45697725 45697723 45697722 45697721 45697720 45697719 45697718 45697694 45697697 45697693 45697695 45697696 45697692 45697691 45697690 45697689 45697688 45697666 45697667 45697662 45697663 45697660 45697664 45697661 45697658 45697659 45697665

```
The number of booked cores is 1024 (1 Parent and 1023 children)
plaintext is: PTPTPT
Random key: (Multiple) and it is used as the start point of the first chain.
After encrypting the plaintext T times (Multiple) the chain end point becomes
: (Multiple)
Ciphertext Challenge: (Multiple)
CT decrypts back to: (Multiple)
output log: (Multiple)
Output size (printf overhead): about 5900 lines, 270000 characters.
error log: (Multiple)
Job log: (Multiple)
source code file:(Multiple)
```

```
job submission script:(Multiple)
object file:(Multiple)
Cores:  1024
usableCores: 1023
PID: (Multiple)
Date: (Multiple)
M=7340032
C=8
W= 1, 2, 3, 4 hours
T=1024
B=48
E=2
H=0
L=0
S=1
R=GPC
notes=(multiple)
```

Observation: All those lightweight runs were successful.

```
45697756
Lead at t=1 core=1, m=0, key1stB_r_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[M-l][ec]__[M-^J][8a]__[M-b][e2]__[M-g][e7]__[^G][07]__[M-^@
   ][80]_
Recovered Key(EP-1):_[r][72]__[K][4b]__[z][7a]__[4][34]__[U][55]__[1][31]_
Chain SP:_[r][72]__[K][4b]__[z][7a]__[4][34]__[U][55]__[1][31]_
Orig CT:_[M-r][f2]__[M-h][e8]__[M-^O][8f]__[M-^A][81]__[S][53]__[^T][14]_
Orig Key:_[r][72]__[K][4b]__[z][7a]__[4][34]__[U][55]__[1][31]_
CT 4 verify:_[M-r][f2]__[M-h][e8]__[M-^O][8f]__[M-^A][81]__[S][53]__[^T
   ][14]_
We have Verification also.
Limits: neednodes=1:ppn=8,nodes=1:ppn=8,walltime=04:00:00
Resources:    cput=02:40:55,mem=316684kb,vmem=2094592kb,walltime=00:20:15

45697757
Lead at t=1 core=1, m=0, key1stB_M_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[n][6e]__[M-w][f7]__[I][49]__[,][2c]__[M-^Z][9a]__[M-)][a9]_
```

```
Recovered Key(EP-1):_[M][4d]__[B][42]__[x][78]__[k][6b]__[7][37]__[5][35]_
Chain SP:_[M][4d]__[B][42]__[x][78]__[k][6b]__[7][37]__[5][35]_
Orig CT:_[^B][02]__[M-?][bf]__[j][6a]__[M-a][e1]__[M-Q][d1]__[M-w][f7]_
Orig Key:_[M][4d]__[B][42]__[x][78]__[k][6b]__[7][37]__[5][35]_
CT 4 verify:_[^B][02]__[M-?][bf]__[j][6a]__[M-a][e1]__[M-Q][d1]__[M-w][f7]
   _
We have Verification also.
Limits: nednodes=1:ppn=8,nodes=1:ppn=8,walltime=04:00:00
Resources:     cput=02:40:59,mem=316172kb,vmem=2094592kb,walltime=00:20:13

45697755
Lead at t=1 core=1, m=0, key1stB_N_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[}][7d]__[J][4a]__[v][76]__[M-/][af]__[^T][14]__[M->][be]_
Recovered Key(EP-1):_[N][4e]__[D][44]__[C][43]__[i][69]__[P][50]__[8][38]_
Chain SP:_[N][4e]__[D][44]__[C][43]__[i][69]__[P][50]__[8][38]_
Orig CT:_[?][3f]__[M-h][e8]__[M-'][a7]__[`][60]__[ ][20]__[M-A][c1]_
Orig Key:_[N][4e]__[D][44]__[C][43]__[i][69]__[P][50]__[8][38]_
CT 4 verify:_[?][3f]__[M-h][e8]__[M-'][a7]__[`][60]__[ ][20]__[M-A][c1]_
We have Verification also.
Limits: nednodes=1:ppn=8,nodes=1:ppn=8,walltime=04:00:00
Resources:     cput=02:41:00,mem=316016kb,vmem=2094592kb,walltime=00:20:14

45697754
Lead at t=1 core=1, m=0, key1stB_A_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[^\][1c]__[C][43]__[M-m][ed]__[:][3a]__[^M][0d]__[M-^G][87]_
Recovered Key(EP-1):_[A][41]__[u][75]__[F][46]__[O][4f]__[R][52]__[b][62]_
Chain SP:_[A][41]__[u][75]__[F][46]__[O][4f]__[R][52]__[b][62]_
Orig CT:_[M-m][ed]__[M-^G][87]__[,][2c]__[M-r][f2]__[M-^L][8c]__["][22]_
Orig Key:_[A][41]__[u][75]__[F][46]__[O][4f]__[R][52]__[b][62]_
CT 4 verify:_[M-m][ed]__[M-^G][87]__[,][2c]__[M-r][f2]__[M-^L][8c]__
   ["][22]_
We have Verification also.
Limits: nednodes=1:ppn=8,nodes=1:ppn=8,walltime=04:00:00
Resources:     cput=02:41:00,mem=316828kb,vmem=2094592kb,walltime=00:20:15

45697753
```

```
Lead at t=1 core=1, m=0, key1stB_c_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[2][32]__[M-^Z][9a]__[M-^T][94]__[M-^Q][91]__[3][33]__[^R][12]_
Recovered Key(EP-1):_[c][63]__[k][6b]__[V][56]__[5][35]__[Z][5a]__[J][4a]_
Chain SP:_[c][63]__[k][6b]__[V][56]__[5][35]__[Z][5a]__[J][4a]_
Orig CT:_[^R][12]__[j][6a]__[M-^F][86]__[k][6b]__[o][6f]__[M-m][ed]_
Orig Key:_[c][63]__[k][6b]__[V][56]__[5][35]__[Z][5a]__[J][4a]_
CT 4 verify:_[^R][12]__[j][6a]__[M-^F][86]__[k][6b]__[o][6f]__[M-m][ed]_
We have Verification also.
Limits: nednodes=1:ppn=8,nodes=1:ppn=8,walltime=04:00:00
Resources:     cput=02:40:48,mem=316460kb,vmem=2094592kb,walltime=00:20:11

45697752
Lead at t=1 core=1, m=0, key1stB_x_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[v][76]__[l][6c]__[M-[][db]__[^T][14]__[0][30]__[M-^Q][91]_
Recovered Key(EP-1):_[x][78]__[N][4e]__[N][4e]__[g][67]__[b][62]__[K][4b]_
Chain SP:_[x][78]__[N][4e]__[N][4e]__[g][67]__[b][62]__[K][4b]_
Orig CT:_[M-^?][ff]__[M-F][c6]__[X][58]__[M-g][e7]__[6][36]__[M-j][ea]_
Orig Key:_[x][78]__[N][4e]__[N][4e]__[g][67]__[b][62]__[K][4b]_
CT 4 verify:_[M-^?][ff]__[M-F][c6]__[X][58]__[M-g][e7]__[6][36]__[M-j][ea]
   _
We have Verification also.
Limits: nednodes=1:ppn=8,nodes=1:ppn=8,walltime=04:00:00
Resources:     cput=02:40:53,mem=317104kb,vmem=2095384kb,walltime=00:20:14

45697751
Lead at t=1 core=1, m=0, key1stB_c_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[R][52]__[=][3d]__[^_][1f]__[X][58]__["][22]__[v][76]_
Recovered Key(EP-1):_[c][63]__[O][4f]__[o][6f]__[d][64]__[2][32]__[j][6a]_
Chain SP:_[c][63]__[O][4f]__[o][6f]__[d][64]__[2][32]__[j][6a]_
Orig CT:_[M-S][d3]__[Y][59]__[M-k][eb]__[R][52]__[M-^C][83]__[^@][00]_
Orig Key:_[c][63]__[O][4f]__[o][6f]__[d][64]__[2][32]__[j][6a]_
CT 4 verify:_[M-S][d3]__[Y][59]__[M-k][eb]__[R][52]__[M-^C][83]__[^@][00]_
We have Verification also.
Limits: nednodes=1:ppn=8,nodes=1:ppn=8,walltime=04:00:00
Resources:     cput=02:40:54,mem=317476kb,vmem=2095516kb,walltime=00:20:11
```

```
45697750
Lead at t=1 core=1, m=0, key1stB_Q_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[M-G][c7]__[g][67]__[q][71]__[M-f][e6]__[|][7c]__[M-c][e3]_
Recovered Key(EP-1):_[Q][51]__[g][67]__[j][6a]__[b][62]__[k][6b]__[L][4c]_
Chain SP:_[Q][51]__[g][67]__[j][6a]__[b][62]__[k][6b]__[L][4c]_
Orig CT:_[S][53]__[M-;][bb]__[M-B][c2]__[^Q][11]__[A][41]__[M-U][d5]_
Orig Key:_[Q][51]__[g][67]__[j][6a]__[b][62]__[k][6b]__[L][4c]_
CT 4 verify:_[S][53]__[M-;][bb]__[M-B][c2]__[^Q][11]__[A][41]__[M-U][d5]_
We have Verification also.
Limits: nodes=1:ppn=8,nodes=1:ppn=8,walltime=04:00:00
Resources:     cput=02:41:02,mem=317336kb,vmem=2095384kb,walltime=00:20:16

45697749
Lead at t=1 core=1, m=0, key1stB_9_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[M-V][d6]__[M-|][fc]__[M-N][ce]__[M-^][de]__[M-^X][98]__[M-E][c5
   ]_
Recovered Key(EP-1):_[9][39]__[4][34]__[t][74]__[U][55]__[0][30]__[m][6d]_
Chain SP:_[9][39]__[4][34]__[t][74]__[U][55]__[0][30]__[m][6d]_
Orig CT:_[7][37]__[I][49]__[x][78]__[l][6c]__[M--][ad]__[M-^H][88]_
Orig Key:_[9][39]__[4][34]__[t][74]__[U][55]__[0][30]__[m][6d]_
CT 4 verify:_[7][37]__[I][49]__[x][78]__[l][6c]__[M--][ad]__[M-^H][88]_
We have Verification also.
Limits: nodes=1:ppn=8,nodes=1:ppn=8,walltime=04:00:00
Resources:     cput=02:41:00,mem=316044kb,vmem=2094592kb,walltime=00:20:14

45697748
Lead at t=1 core=1, m=0, key1stB_Z_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[M-4][b4]__[L][4c]__[^D][04]__[M-^Z][9a]__[[][5b]__[#][23]_
Recovered Key(EP-1):_[Z][5a]__[y][79]__[n][6e]__[w][77]__[5][35]__[7][37]_
Chain SP:_[Z][5a]__[y][79]__[n][6e]__[w][77]__[5][35]__[7][37]_
Orig CT:_[^^][1e]__[^D][04]__[M-J][ca]__[M-^L][8c]__[0][30]__[M-;][bb]_
Orig Key:_[Z][5a]__[y][79]__[n][6e]__[w][77]__[5][35]__[7][37]_
CT 4 verify:_[^^][1e]__[^D][04]__[M-J][ca]__[M-^L][8c]__[0][30]__[M-;][bb]
   _
```

```
We have Verification also.
Limits: neednodes=1:ppn=8,nodes=1:ppn=8,walltime=04:00:00
Resources:     cput=02:40:55,mem=315720kb,vmem=2094592kb,walltime=00:20:14

45697727
Lead at t=1 core=1, m=0, key1stB_A_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[^\][1c]__[M-]][dd]__[M-G][c7]__[M-0][b0]__["][22]__[V][56]_
Recovered Key(EP-1):_[A][41]__[y][79]__[y][79]__[l][6c]__[o][6f]__[W][57]_
Chain SP:_[A][41]__[y][79]__[y][79]__[l][6c]__[o][6f]__[W][57]_
Orig CT:_[s][73]__[F][46]__[a][61]__[e][65]__[M-g][e7]__[M-^?][ff]_
Orig Key:_[A][41]__[y][79]__[y][79]__[l][6c]__[o][6f]__[W][57]_
CT 4 verify:_[s][73]__[F][46]__[a][61]__[e][65]__[M-g][e7]__[M-^?][ff]_
We have Verification also.
Limits: neednodes=1:ppn=8,nodes=1:ppn=8,walltime=03:00:00
Resources:     cput=02:41:07,mem=316240kb,vmem=2094592kb,walltime=00:20:16

45697726
Lead at t=1 core=1, m=0, key1stB_x_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[M-I][c9]__[a][61]__[M-1][b1]__[@][40]__[M-.][ae]__[f][66]_
Recovered Key(EP-1):_[x][78]__[D][44]__[P][50]__[e][65]__[4][34]__[i][69]_
Chain SP:_[x][78]__[D][44]__[P][50]__[e][65]__[4][34]__[i][69]_
Orig CT:_[L][4c]__[^][5e]__[.][2e]__[&][26]__[c][63]__[‘][60]_
Orig Key:_[x][78]__[D][44]__[P][50]__[e][65]__[4][34]__[i][69]_
CT 4 verify:_[L][4c]__[^][5e]__[.][2e]__[&][26]__[c][63]__[‘][60]_
We have Verification also.
Limits: neednodes=1:ppn=8,nodes=1:ppn=8,walltime=03:00:00
Resources:     cput=02:40:57,mem=316228kb,vmem=2094592kb,walltime=00:20:13

45697724
Lead at t=1 core=1, m=0, key1stB_I_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[^D][04]__[M-}][fd]__[^U][15]__[}][7d]__[M-]][dd]__["][22]_
Recovered Key(EP-1):_[I][49]__[A][41]__[X][58]__[d][64]__[E][45]__[M][4d]_
Chain SP:_[I][49]__[A][41]__[X][58]__[d][64]__[E][45]__[M][4d]_
Orig CT:_()][29]__[M-^N][8e]__[<][3c]__[M-o][ef]__[q][71]__[@][40]_
Orig Key:_[I][49]__[A][41]__[X][58]__[d][64]__[E][45]__[M][4d]_
```

```
CT 4 verify:_()][29]__[M-^N][8e]__[<][3c]__[M-o][ef]__[q][71]__[@][40]_
We have Verification also.
Limits: nednodes=1:ppn=8,nodes=1:ppn=8,walltime=03:00:00
Resources:     cput=02:40:58,mem=316816kb,vmem=2094592kb,walltime=00:20:14

45697725
Lead at t=1 core=1, m=0, key1stB_Q_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[M-L][cc]__[^V][16]__[I][49]__[p][70]__[M-5][b5]__[M-/][af]_
Recovered Key(EP-1):_[Q][51]__[e][65]__[C][43]__[p][70]__[p][70]__[N][4e]_
Chain SP:_[Q][51]__[e][65]__[C][43]__[p][70]__[p][70]__[N][4e]_
Orig CT:_[M-q][f1]__[F][46]__[p][70]__[M-^\][9c]__[M-6][b6]__[M-F][c6]_
Orig Key:_[Q][51]__[e][65]__[C][43]__[p][70]__[p][70]__[N][4e]_
CT 4 verify:_[M-q][f1]__[F][46]__[p][70]__[M-^\][9c]__[M-6][b6]__[M-F][c6]
   _
We have Verification also.
Limits: nednodes=1:ppn=8,nodes=1:ppn=8,walltime=03:00:00
Resources:     cput=02:40:56,mem=315676kb,vmem=2094592kb,walltime=00:20:13

45697723
Lead at t=1 core=1, m=0, key1stB_M_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[T][54]__[#][23]__[M-^A][81]__[M-^A][81]__[M-[][db]__[M-^\][9c]_
Recovered Key(EP-1):_[M][4d]__[H][48]__[m][6d]__[K][4b]__[j][6a]__[k][6b]_
Chain SP:_[M][4d]__[H][48]__[m][6d]__[K][4b]__[j][6a]__[k][6b]_
Orig CT:_[f][66]__[^K][0b]__[J][4a]__[M-^O][8f]__[M-k][eb]__[}][7d]_
Orig Key:_[M][4d]__[H][48]__[m][6d]__[K][4b]__[j][6a]__[k][6b]_
CT 4 verify:_[f][66]__[^K][0b]__[J][4a]__[M-^O][8f]__[M-k][eb]__[}][7d]_
We have Verification also.
Limits: nednodes=1:ppn=8,nodes=1:ppn=8,walltime=03:00:00
Resources:     cput=02:41:01,mem=315340kb,vmem=2095616kb,walltime=00:20:14

45697722
Lead at t=1 core=1, m=0, key1stB_Z_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[M-K][cb]__[R][52]__[M-q][f1]__[M-^\][9c]__[M-[][db]__[M-W][d7]_
Recovered Key(EP-1):_[Z][5a]__[t][74]__[g][67]__[1][31]__[t][74]__[l][6c]_
Chain SP:_[Z][5a]__[t][74]__[g][67]__[1][31]__[t][74]__[l][6c]_
```

```
Orig CT:_()][29]__[B][42]__[M-^X][98]__[^?][7f]__[M-^A][81]__[ ][20]_
Orig Key:_[Z][5a]__[t][74]__[g][67]__[1][31]__[t][74]__[l][6c]_
CT 4 verify:_()][29]__[B][42]__[M-^X][98]__[^?][7f]__[M-^A][81]__[ ][20]_
We have Verification also.
Limits: nednodes=1:ppn=8,nodes=1:ppn=8,walltime=03:00:00
Resources:     cput=02:40:56,mem=315856kb,vmem=2094592kb,walltime=00:20:13

45697721
Lead at t=1 core=1, m=0, key1stB_I_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[M-O][cf]__[M-!][a1]__[M-‘][e0]__[M-^M][8d]__[M-S][d3]__[’][27]_
Recovered Key(EP-1):_[I][49]__[Z][5a]__[a][61]__[Q][51]__[G][47]__[x][78]_
Chain SP:_[I][49]__[Z][5a]__[a][61]__[Q][51]__[G][47]__[x][78]_
Orig CT:_[s][73]__[M-_][df]__[M-n][ee]__[M-^F][86]__[M-^V][96]__[M-b][e2]_
Orig Key:_[I][49]__[Z][5a]__[a][61]__[Q][51]__[G][47]__[x][78]_
CT 4 verify:_[s][73]__[M-_][df]__[M-n][ee]__[M-^F][86]__[M-^V][96]__[M-b][
   e2]_
We have Verification also.
Limits: nednodes=1:ppn=8,nodes=1:ppn=8,walltime=03:00:00
Resources:     cput=02:41:02,mem=316632kb,vmem=2094592kb,walltime=00:20:16

45697720
Lead at t=1 core=1, m=0, key1stB_4_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[M-i][e9]__[k][6b]__[Y][59]__[M-f][e6]__[M-\][dc]__[^^][1e]_
Recovered Key(EP-1):_[4][34]__[t][74]__[M][4d]__[s][73]__[P][50]__[7][37]_
Chain SP:_[4][34]__[t][74]__[M][4d]__[s][73]__[P][50]__[7][37]_
Orig CT:_[M-^C][83]__[8][38]__[|][7c]__[M][4d]__[M-_][df]__[M-Y][d9]_
Orig Key:_[4][34]__[t][74]__[M][4d]__[s][73]__[P][50]__[7][37]_
CT 4 verify:_[M-^C][83]__[8][38]__[|][7c]__[M][4d]__[M-_][df]__[M-Y][d9]_
We have Verification also.
Limits: nednodes=1:ppn=8,nodes=1:ppn=8,walltime=03:00:00
Resources:     cput=02:41:03,mem=315604kb,vmem=2094592kb,walltime=00:20:16

45697719
Lead at t=1 core=1, m=0, key1stB_h_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[3][33]__[m][6d]__[M-\][dc]__[o][6f]__[t][74]__[5][35]_
```

```
Recovered Key(EP-1):_[h][68]__[B][42]__[f][66]__[X][58]__[C][43]__[I][49]_
Chain SP:_[h][68]__[B][42]__[f][66]__[X][58]__[C][43]__[I][49]_
Orig CT:_[M-^A][81]__[M][4d]__[^Q][11]__[M-p][f0]__[^Y][19]__[M-^_][9f]_
Orig Key:_[h][68]__[B][42]__[f][66]__[X][58]__[C][43]__[I][49]_
CT 4 verify:_[M-^A][81]__[M][4d]__[^Q][11]__[M-p][f0]__[^Y][19]__[M-^_][9f
   ]_
We have Verification also.
Limits: nednodes=1:ppn=8,nodes=1:ppn=8,walltime=03:00:00
Resources:     cput=02:40:59,mem=317372kb,vmem=2095384kb,walltime=00:20:15

45697718
][0a]__[M-[][db]__[^_][1f]_
We have Verification also.
Limits: nednodes=1:ppn=8,nodes=1:ppn=8,walltime=03:00:00
Resources:     cput=02:40:59,mem=317352kb,vmem=2095516kb,walltime=00:20:13

45697694
Lead at t=1 core=1, m=0, key1stB_t_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[^O][0f]__[M-3][b3]__["][22]__[M-4][b4]__[l][6c]__[f][66]_
Recovered Key(EP-1):_[t][74]__[E][45]__[t][74]__[K][4b]__[J][4a]__[G][47]_
Chain SP:_[t][74]__[E][45]__[t][74]__[K][4b]__[J][4a]__[G][47]_
Orig CT:_[^][5e]__[M-L][cc]__[M-o][ef]__[-][2d]__[#][23]__[M-L][cc]_
Orig Key:_[t][74]__[E][45]__[t][74]__[K][4b]__[J][4a]__[G][47]_
CT 4 verify:_[^][5e]__[M-L][cc]__[M-o][ef]__[-][2d]__[#][23]__[M-L][cc]_
We have Verification also.
Limits: nednodes=1:ppn=8,nodes=1:ppn=8,walltime=02:00:00
Resources:     cput=02:40:59,mem=316844kb,vmem=2094592kb,walltime=00:20:15

45697697
Lead at t=1 core=1, m=0, key1stB_c_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[*][2a]__[M-$][a4]__[^G][07]__[M-G][c7]__[M-~][fe]__[^P][10]_
Recovered Key(EP-1):_[c][63]__[j][6a]__[5][35]__[G][47]__[Q][51]__[q][71]_
Chain SP:_[c][63]__[j][6a]__[5][35]__[G][47]__[Q][51]__[q][71]_
Orig CT:_[1][31]__[J][4a]__[M-c][e3]__[M-h][e8]__[n][6e]__[C][43]_
Orig Key:_[c][63]__[j][6a]__[5][35]__[G][47]__[Q][51]__[q][71]_
CT 4 verify:_[1][31]__[J][4a]__[M-c][e3]__[M-h][e8]__[n][6e]__[C][43]_
```

```
We have Verification also.
Limits: neednodes=1:ppn=8,nodes=1:ppn=8,walltime=02:00:00
Resources:     cput=02:41:00,mem=317372kb,vmem=2095516kb,walltime=00:20:16

45697693
Lead at t=1 core=1, m=0, key1stB_W_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[^L][0c]__[<][3c]__[M-^N][8e]__[M-^R][92]__[M-X][d8]__[M-e][e5]_
Recovered Key(EP-1):_[W][57]__[0][30]__[i][69]__[d][64]__[2][32]__[i][69]_
Chain SP:_[W][57]__[0][30]__[i][69]__[d][64]__[2][32]__[i][69]_
Orig CT:_[m][6d]__[M-^R][92]__[M-^_][9f]__[M-^Y][99]__[M-,][ac]__[u][75]_
Orig Key:_[W][57]__[0][30]__[i][69]__[d][64]__[2][32]__[i][69]_
CT 4 verify:_[m][6d]__[M-^R][92]__[M-^_][9f]__[M-^Y][99]__[M-,][ac]__[u
   ][75]_
We have Verification also.
Limits: neednodes=1:ppn=8,nodes=1:ppn=8,walltime=02:00:00
Resources:     cput=02:40:58,mem=316296kb,vmem=2094592kb,walltime=00:20:14

45697695
Lead at t=1 core=1, m=0, key1stB_t_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[^_][1f]__[M-0][b0]__[M-(][a8]__[E][45]__[M-]][dd]__[V][56]_
Recovered Key(EP-1):_[t][74]__[v][76]__[1][31]__[K][4b]__[S][53]__[b][62]_
Chain SP:_[t][74]__[v][76]__[1][31]__[K][4b]__[S][53]__[b][62]_
Orig CT:_[?][3f]__[M-j][ea]__[M-(][a8]__[M-‘][e0]__[u][75]__[y][79]_
Orig Key:_[t][74]__[v][76]__[1][31]__[K][4b]__[S][53]__[b][62]_
CT 4 verify:_[?][3f]__[M-j][ea]__[M-(][a8]__[M-‘][e0]__[u][75]__[y][79]_
We have Verification also.
Limits: neednodes=1:ppn=8,nodes=1:ppn=8,walltime=02:00:00
Resources:     cput=02:41:03,mem=316360kb,vmem=2094592kb,walltime=00:20:14

45697696
Lead at t=1 core=1, m=0, key1stB_u_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[2][32]__[M-4][b4]__[M-V][d6]__[M-+][ab]__[U][55]__[M-^X][98]_
Recovered Key(EP-1):_[u][75]__[P][50]__[v][76]__[n][6e]__[k][6b]__[L][4c]_
Chain SP:_[u][75]__[P][50]__[v][76]__[n][6e]__[k][6b]__[L][4c]_
Orig CT:_[M-$][a4]__[M-0][b0]__[M-T][d4]__[^A][01]__[M-z][fa]__[M-S][d3]_
```

```
Orig Key:_[u][75]__[P][50]__[v][76]__[n][6e]__[k][6b]__[L][4c]_
CT 4 verify:_[M-$][a4]__[M-0][b0]__[M-T][d4]__[^A][01]__[M-z][fa]__[M-S][
   d3]_
We have Verification also.
Limits: neednodes=1:ppn=8,nodes=1:ppn=8,walltime=02:00:00
Resources:     cput=02:40:59,mem=315620kb,vmem=2094592kb,walltime=00:20:13

45697692
Lead at t=1 core=1, m=0, key1stB_q_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[M-h][e8]__[M-v][f6]__[    ][09]__[^^][1e]__[ ][20]__[M-{][fb]_
Recovered Key(EP-1):_[q][71]__[h][68]__[t][74]__[Z][5a]__[R][52]__[A][41]_
Chain SP:_[q][71]__[h][68]__[t][74]__[Z][5a]__[R][52]__[A][41]_
Orig CT:_[&][26]__[M-#][a3]__[2][32]__[@][40]__[N][4e]__[M-:][ba]_
Orig Key:_[q][71]__[h][68]__[t][74]__[Z][5a]__[R][52]__[A][41]_
CT 4 verify:_[&][26]__[M-#][a3]__[2][32]__[@][40]__[N][4e]__[M-:][ba]_
We have Verification also.
Limits: neednodes=1:ppn=8,nodes=1:ppn=8,walltime=02:00:00
Resources:     cput=02:40:50,mem=315268kb,vmem=2094592kb,walltime=00:20:13

45697691
Lead at t=1 core=1, m=0, key1stB_A_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[6][36]__[M-I][c9]__[M-C][c3]__['][27]__[l][6c]__[^][5e]_
Recovered Key(EP-1):_[A][41]__[2][32]__[w][77]__[k][6b]__[k][6b]__[J][4a]_
Chain SP:_[A][41]__[2][32]__[w][77]__[k][6b]__[k][6b]__[J][4a]_
Orig CT:_[M-J][ca]__[M-I][c9]__[^E][05]__[5][35]__[M-^V][96]__[j][6a]_
Orig Key:_[A][41]__[2][32]__[w][77]__[k][6b]__[k][6b]__[J][4a]_
CT 4 verify:_[M-J][ca]__[M-I][c9]__[^E][05]__[5][35]__[M-^V][96]__[j][6a]_
We have Verification also.
Limits: neednodes=1:ppn=8,nodes=1:ppn=8,walltime=02:00:00
Resources:     cput=02:40:53,mem=315948kb,vmem=2094592kb,walltime=00:20:13

45697690
Lead at t=1 core=1, m=0, key1stB_N_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[0][30]__[@][40]__[T][54]__[M-6][b6]__[#][23]__[p][70]_
Recovered Key(EP-1):_[N][4e]__[R][52]__[i][69]__[d][64]__[l][6c]__[G][47]_
```

```
Chain SP:_[N][4e]__[R][52]__[i][69]__[d][64]__[l][6c]__[G][47]_
Orig CT:_[M-^U][95]__[M-*][aa]__[M-9][b9]__[L][4c]__[M-^H][88]__[[][5b]_
Orig Key:_[N][4e]__[R][52]__[i][69]__[d][64]__[l][6c]__[G][47]_
CT 4 verify:_[M-^U][95]__[M-*][aa]__[M-9][b9]__[L][4c]__[M-^H][88]__[[][5b
   ]_
We have Verification also.
Limits: neednodes=1:ppn=8,nodes=1:ppn=8,walltime=02:00:00
Resources:     cput=02:40:52,mem=316108kb,vmem=2094592kb,walltime=00:20:12

45697689
Lead at t=1 core=1, m=0, key1stB_1_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[M-g][e7]__[M-^A][81]__[M-^Q][91]__[E][45]__[y][79]__[M-O][cf]_
Recovered Key(EP-1):_[1][31]__[M][4d]__[a][61]__[W][57]__[N][4e]__[c][63]_
Chain SP:_[1][31]__[M][4d]__[a][61]__[W][57]__[N][4e]__[c][63]_
Orig CT:_[n][6e]__[ ][20]__[5][35]__[M->][be]__[M-e][e5]__[M-R][d2]_
Orig Key:_[1][31]__[M][4d]__[a][61]__[W][57]__[N][4e]__[c][63]_
CT 4 verify:_[n][6e]__[ ][20]__[5][35]__[M->][be]__[M-e][e5]__[M-R][d2]_
We have Verification also.
Limits: neednodes=1:ppn=8,nodes=1:ppn=8,walltime=02:00:00
Resources:     cput=02:41:09,mem=315912kb,vmem=2094592kb,walltime=00:20:16

45697688
Lead at t=1 core=1, m=0, key1stB_P_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[g][67]__[M-_][df]__[~][7e]__[O][4f]__[^E][05]__[R][52]_
Recovered Key(EP-1):_[P][50]__[X][58]__[Z][5a]__[g][67]__[g][67]__[j][6a]_
Chain SP:_[P][50]__[X][58]__[Z][5a]__[g][67]__[g][67]__[j][6a]_
Orig CT:_[V][56]__[M-`][e0]__[M-^A][81]__[M-(][a8]__[M-^K][8b]__[Y][59]_
Orig Key:_[P][50]__[X][58]__[Z][5a]__[g][67]__[g][67]__[j][6a]_
CT 4 verify:_[V][56]__[M-`][e0]__[M-^A][81]__[M-(][a8]__[M-^K][8b]__[Y
   ][59]_
We have Verification also.
Limits: neednodes=1:ppn=8,nodes=1:ppn=8,walltime=02:00:00
Resources:     cput=02:40:48,mem=315516kb,vmem=2094592kb,walltime=00:20:12

45697666
Lead at t=1 core=1, m=0, key1stB_c_
```

```
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[M-=][bd]__[M-^M][8d]__[M-B][c2]__[M-^M][8d]__[^S][13]__[+][2b]_
Recovered Key(EP-1):_[c][63]__[6][36]__[S][53]__[n][6e]__[S][53]__[Y][59]_
Chain SP:_[c][63]__[6][36]__[S][53]__[n][6e]__[S][53]__[Y][59]_
Orig CT:_[!][21]__[M-H][c8]__[M-^X][98]__[M-F][c6]__[M-^J][8a]__[;][3b]_
Orig Key:_[c][63]__[6][36]__[S][53]__[n][6e]__[S][53]__[Y][59]_
CT 4 verify:_[!][21]__[M-H][c8]__[M-^X][98]__[M-F][c6]__[M-^J][8a]__[;][3b
   ]_
We have Verification also.
Limits: nednodes=1:ppn=8,nodes=1:ppn=8,walltime=01:00:00
Resources:     cput=02:40:54,mem=316720kb,vmem=2095556kb,walltime=00:20:14

45697667
Lead at t=1 core=1, m=0, key1stB_B_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[?][3f]__[M-g][e7]__[M-3][b3]__[^P][10]__[M-s][f3]__[?][3f]_
Recovered Key(EP-1):_[B][42]__[k][6b]__[A][41]__[4][34]__[C][43]__[D][44]_
Chain SP:_[B][42]__[k][6b]__[A][41]__[4][34]__[C][43]__[D][44]_
Orig CT:_[M-s][f3]__[M-v][f6]__[M-^I][89]__[^Z][1a]__[e][65]__[M-$][a4]_
Orig Key:_[B][42]__[k][6b]__[A][41]__[4][34]__[C][43]__[D][44]_
CT 4 verify:_[M-s][f3]__[M-v][f6]__[M-^I][89]__[^Z][1a]__[e][65]__[M-$][a4
   ]_
We have Verification also.
Limits: nednodes=1:ppn=8,nodes=1:ppn=8,walltime=01:00:00
Resources:     cput=02:40:59,mem=316128kb,vmem=2094592kb,walltime=00:20:13

45697662
Lead at t=1 core=1, m=0, key1stB_t_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[^P][10]__[C][43]__[M-R][d2]__[M-^C][83]__[c][63]__[M-T][d4]_
Recovered Key(EP-1):_[t][74]__[Z][5a]__[O][4f]__[w][77]__[Y][59]__[x][78]_
Chain SP:_[t][74]__[Z][5a]__[O][4f]__[w][77]__[Y][59]__[x][78]_
Orig CT:_["][22]__[@][40]__[M-i][e9]__[M-2][b2]__[M-n][ee]__[^][5e]_
Orig Key:_[t][74]__[Z][5a]__[O][4f]__[w][77]__[Y][59]__[x][78]_
CT 4 verify:_["][22]__[@][40]__[M-i][e9]__[M-2][b2]__[M-n][ee]__[^][5e]_
We have Verification also.
Limits: nednodes=1:ppn=8,nodes=1:ppn=8,walltime=01:00:00
Resources:     cput=02:41:19,mem=316632kb,vmem=2094592kb,walltime=00:20:20
```

```
45697663
Lead at t=1 core=1, m=0, key1stB_y_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[B][42]__[V][56]__[M-(][a8]__[M-P][d0]__[+][2b]__[.][2e]_
Recovered Key(EP-1):_[y][79]__[J][4a]__[w][77]__[R][52]__[l][6c]__[3][33]_
Chain SP:_[y][79]__[J][4a]__[w][77]__[R][52]__[l][6c]__[3][33]_
Orig CT:_[,][2c]__[M-^O][8f]__[=][3d]__[^X][18]__[M-,][ac]__[+][2b]_
Orig Key:_[y][79]__[J][4a]__[w][77]__[R][52]__[l][6c]__[3][33]_
CT 4 verify:_[,][2c]__[M-^O][8f]__[=][3d]__[^X][18]__[M-,][ac]__[+][2b]_
We have Verification also.
Limits: nednodes=1:ppn=8,nodes=1:ppn=8,walltime=01:00:00
Resources:     cput=02:41:06,mem=316128kb,vmem=2094592kb,walltime=00:20:18

45697660
Lead at t=1 core=1, m=0, key1stB_P_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[M-%][a5]__[M-^_][9f]__[M-j][ea]__[9][39]__[G][47]__[M-h][e8]_
Recovered Key(EP-1):_[P][50]__[q][71]__[R][52]__[u][75]__[r][72]__[q][71]_
Chain SP:_[P][50]__[q][71]__[R][52]__[u][75]__[r][72]__[q][71]_
Orig CT:_[M-.][ae]__[W][57]__[M--][ad]__[M-@][c0]__[X][58]__[g][67]_
Orig Key:_[P][50]__[q][71]__[R][52]__[u][75]__[r][72]__[q][71]_
CT 4 verify:_[M-.][ae]__[W][57]__[M--][ad]__[M-@][c0]__[X][58]__[g][67]_
We have Verification also.
Limits: nednodes=1:ppn=8,nodes=1:ppn=8,walltime=01:00:00
Resources:     cput=02:41:05,mem=316120kb,vmem=2094592kb,walltime=00:20:17

45697664
Lead at t=1 core=1, m=0, key1stB_6_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[(][28]__[g][67]__[M-j][ea]__[]][5d]__[M-^@][80]__[k][6b]_
Recovered Key(EP-1):_[6][36]__[c][63]__[8][38]__[O][4f]__[n][6e]__[M][4d]_
Chain SP:_[6][36]__[c][63]__[8][38]__[O][4f]__[n][6e]__[M][4d]_
Orig CT:_[M-^B][82]__[^_][1f]__[M-'][a7]__[;][3b]__[U][55]__[M-E][c5]_
Orig Key:_[6][36]__[c][63]__[8][38]__[O][4f]__[n][6e]__[M][4d]_
CT 4 verify:_[M-^B][82]__[^_][1f]__[M-'][a7]__[;][3b]__[U][55]__[M-E][c5]_
We have Verification also.
Limits: nednodes=1:ppn=8,nodes=1:ppn=8,walltime=01:00:00
```

```
Resources:     cput=02:41:01,mem=313100kb,vmem=2095380kb,walltime=00:20:16

45697661
][0a]_
We have Verification also.
Limits: nednodes=1:ppn=8,nodes=1:ppn=8,walltime=01:00:00
Resources:     cput=02:41:00,mem=317444kb,vmem=2095516kb,walltime=00:20:16

45697658
Lead at t=1 core=1, m=0, key1stB_J_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[7][37]__[p][70]__[M-_][df]__[7][37]__[S][53]__[M-^V][96]_
Recovered Key(EP-1):_[J][4a]__[N][4e]__[Q][51]__[2][32]__[x][78]__[c][63]_
Chain SP:_[J][4a]__[N][4e]__[Q][51]__[2][32]__[x][78]__[c][63]_
Orig CT:_[M-^L][8c]__[M-^R][92]__[Y][59]__[0][30]__[M-^O][8f]__[^G][07]_
Orig Key:_[J][4a]__[N][4e]__[Q][51]__[2][32]__[x][78]__[c][63]_
CT 4 verify:_[M-^L][8c]__[M-^R][92]__[Y][59]__[0][30]__[M-^O][8f]__[^G
   ][07]_
We have Verification also.
Limits: nednodes=1:ppn=8,nodes=1:ppn=8,walltime=01:00:00
Resources:     cput=02:40:54,mem=316476kb,vmem=2094592kb,walltime=00:20:13

45697659
Lead at t=1 core=1, m=0, key1stB_C_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[M-E][c5]__[7][37]__[M-P][d0]__[^[][1b]__[M-x][f8]__[P][50]_
Recovered Key(EP-1):_[C][43]__[n][6e]__[4][34]__[t][74]__[e][65]__[6][36]_
Chain SP:_[C][43]__[n][6e]__[4][34]__[t][74]__[e][65]__[6][36]_
Orig CT:_[{][7b]__[>][3e]__[/][2f]__[M-^J][8a]__[M-/][af]__[m][6d]_
Orig Key:_[C][43]__[n][6e]__[4][34]__[t][74]__[e][65]__[6][36]_
CT 4 verify:_[{][7b]__[>][3e]__[/][2f]__[M-^J][8a]__[M-/][af]__[m][6d]_
We have Verification also.
Limits: nednodes=1:ppn=8,nodes=1:ppn=8,walltime=01:00:00
Resources:     cput=02:40:58,mem=315572kb,vmem=2094592kb,walltime=00:20:13

45697665
Lead at t=1 core=1, m=0, key1stB_g_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
```

```
Found CT:_[M-^D][84]__[f][66]__[-][2d]__[M-a][e1]__[c][63]__[^_][1f]_
Recovered Key(EP-1):_[g][67]__[X][58]__[D][44]__[C][43]__[x][78]__[H][48]_
Chain SP:_[g][67]__[X][58]__[D][44]__[C][43]__[x][78]__[H][48]_
Orig CT:_[M-^P][90]__[M-)][a9]__[^][5e]__[^O][0f]__[M-]][dd]__[M-J][ca]_
Orig Key:_[g][67]__[X][58]__[D][44]__[C][43]__[x][78]__[H][48]_
CT 4 verify:_[M-^P][90]__[M-)][a9]__[^][5e]__[^O][0f]__[M-]][dd]__[M-J][ca
   ]_
We have Verification also.
Limits: nodes=1:ppn=8,walltime=01:00:00
Resources:    cput=02:40:48,mem=316124kb,vmem=2094592kb,walltime=00:20:12
```

A.1.7 32c 32505856m

Job ID (45697777 45697776 45697775 45697774 45697773 45697772 45697771 45697770 45697769 45697768 45697747 45697746 45697745 45697744 45697743 45697742 45697741 45697739 45697740 45697738 45697717 45697715 45697716 45697714 45697713 45697712 45697711 45697710 45697709 45697708 45697686 45697687 45697685 45697684 45697683 45697682 45697680 45697681 45697679 45697678)

```
The number of booked cores is 32 (1 Parent and 31 children)
plaintext is: PTPTPT
Random key: (Multiple) and it is used as the start point of the first
   chain.
After encrypting the plaintext T times (1024) the chain end point becomes
   : (Multiple)
Ciphertext Challenge: (Multiple)
CT decrypts back to: (Multiple)
output log:
Output size (printf overhead): about 25000 lines, 1150000 characters.
error log: (Multiple)
Job log: (Multiple)
source code file:(Multiple)
job submission script:(Multiple)
object file:(Multiple)
Cores:32
usableCores:31
```

```
PID:(Multiple)
Date: (Multiple)
M=32505856
C=32
W=32505856m
T=1024
B=48
E=2
H=0
L=0
S=1
R=GPC
notes=(Multiple)
```

Observation: All runs were successful.

```
45697777
Lead at t=1 core=1, m=0, key1stB_G_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[M-s][f3]__[M--][ad]__[u][75]__[^E][05]__[y][79]__[M-^C][83]_
Recovered Key(EP-1):_[G][47]__[9][39]__[U][55]__[8][38]__[W][57]__[L][4c]_
Chain SP:_[G][47]__[9][39]__[U][55]__[8][38]__[W][57]__[L][4c]_
Orig CT:_[X][58]__[M-x][f8]__[k][6b]__[5][35]__[M-h][e8]__[M-Q][d1]_
Orig Key:_[G][47]__[9][39]__[U][55]__[8][38]__[W][57]__[L][4c]_
CT 4 verify:_[X][58]__[M-x][f8]__[k][6b]__[5][35]__[M-h][e8]__[M-Q][d1]_
We have Verification also.
Limits: nednodes=4:ppn=8,nodes=4:ppn=8,walltime=04:00:00
Resources:     cput=10:45:31,mem=1443792kb,vmem=8481168kb,walltime=00:20:20

45697776
Lead at t=1 core=1, m=0, key1stB_T_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[q][71]__[M-L][cc]__[V][56]__[M-^_][9f]__[^P][10]__[W][57]_
Recovered Key(EP-1):_[T][54]__[s][73]__[X][58]__[i][69]__[v][76]__[0][30]_
Chain SP:_[T][54]__[s][73]__[X][58]__[i][69]__[v][76]__[0][30]_
Orig CT:_[U][55]__[M-^B][82]__[M-M][cd]__[M-^V][96]__[3][33]__[I][49]_
Orig Key:_[T][54]__[s][73]__[X][58]__[i][69]__[v][76]__[0][30]_
CT 4 verify:_[U][55]__[M-^B][82]__[M-M][cd]__[M-^V][96]__[3][33]__[I][49]_
```

```
We have Verification also.
Limits: neednodes=4:ppn=8,nodes=4:ppn=8,walltime=04:00:00
Resources:     cput=10:43:53,mem=1445884kb,vmem=8480472kb,walltime=00:20:16

45697775
Lead at t=1 core=1, m=0, key1stB_i_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[+][2b]__[B][42]__[h][68]__[^O][0f]__[O][4f]__[M-G][c7]_
Recovered Key(EP-1):_[i][69]__[t][74]__[k][6b]__[J][4a]__[B][42]__[G][47]_
Chain SP:_[i][69]__[t][74]__[k][6b]__[J][4a]__[B][42]__[G][47]_
Orig CT:_()][29]__[9][39]__[Q][51]__[M-C][c3]__[M-5][b5]__[M-z][fa]_
Orig Key:_[i][69]__[t][74]__[k][6b]__[J][4a]__[B][42]__[G][47]_
CT 4 verify:_()][29]__[9][39]__[Q][51]__[M-C][c3]__[M-5][b5]__[M-z][fa]_
We have Verification also.
Limits: neednodes=4:ppn=8,nodes=4:ppn=8,walltime=04:00:00
Resources:     cput=10:45:08,mem=1443476kb,vmem=8481892kb,walltime=00:20:19

45697774
Lead at t=1 core=1, m=0, key1stB_5_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[6][36]__[2][32]__[s][73]__[M-k][eb]__[M--][ad]__[K][4b]_
Recovered Key(EP-1):_[5][35]__[r][72]__[K][4b]__[J][4a]__[z][7a]__[j][6a]_
Chain SP:_[5][35]__[r][72]__[K][4b]__[J][4a]__[z][7a]__[j][6a]_
Orig CT:_[c][63]__[^M][0d]__[^X][18]__[M-%][a5]__[(][28]__[P][50]_
Orig Key:_[5][35]__[r][72]__[K][4b]__[J][4a]__[z][7a]__[j][6a]_
CT 4 verify:_[c][63]__[^M][0d]__[^X][18]__[M-%][a5]__[(][28]__[P][50]_
We have Verification also.
Limits: neednodes=4:ppn=8,nodes=4:ppn=8,walltime=04:00:00
Resources:     cput=10:43:40,mem=1437824kb,vmem=8480476kb,walltime=00:20:18

45697773
Lead at t=1 core=1, m=0, key1stB_U_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[|][7c]__[Z][5a]__[Y][59]__[M-q][f1]__[M-^K][8b]__[M-}][fd]_
Recovered Key(EP-1):_[U][55]__[u][75]__[C][43]__[a][61]__[s][73]__[O][4f]_
Chain SP:_[U][55]__[u][75]__[C][43]__[a][61]__[s][73]__[O][4f]_
Orig CT:_[D][44]__[I][49]__[M-f][e6]__[(][28]__[^T][14]__[M-h][e8]_
Orig Key:_[U][55]__[u][75]__[C][43]__[a][61]__[s][73]__[O][4f]_
```

```
CT 4 verify:_[D][44]__[I][49]__[M-f][e6]__[(][28]__[^T][14]__[M-h][e8]_
We have Verification also.
Limits: nednodes=4:ppn=8,nodes=4:ppn=8,walltime=04:00:00
Resources:     cput=10:44:57,mem=1442768kb,vmem=8479332kb,walltime=00:20:17

45697772
Lead at t=1 core=1, m=0, key1stB_c_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[M-^Z][9a]__[@][40]__[B][42]__[M-+][ab]__[M-{][fb]__[M-^@][80]_
Recovered Key(EP-1):_[c][63]__[W][57]__[K][4b]__[O][4f]__[k][6b]__[q][71]_
Chain SP:_[c][63]__[W][57]__[K][4b]__[O][4f]__[k][6b]__[q][71]_
Orig CT:_[=][3d]__[M-*][aa]__[M-^V][96]__[C][43]__[y][79]__[(][28]_
Orig Key:_[c][63]__[W][57]__[K][4b]__[O][4f]__[k][6b]__[q][71]_
CT 4 verify:_[=][3d]__[M-*][aa]__[M-^V][96]__[C][43]__[y][79]__[(][28]_
We have Verification also.
Limits: nednodes=4:ppn=8,nodes=4:ppn=8,walltime=04:00:00
Resources:     cput=10:43:43,mem=1446796kb,vmem=8480472kb,walltime=00:20:17

45697771
Lead at t=1 core=1, m=0, key1stB_X_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[^R][12]__[M-Z][da]__[M-:][ba]__[M-^P][90]__[M-o][ef]__[V][56]_
Recovered Key(EP-1):_[X][58]__[R][52]__[F][46]__[m][6d]__[B][42]__[r][72]_
Chain SP:_[X][58]__[R][52]__[F][46]__[m][6d]__[B][42]__[r][72]_
Orig CT:_[M-Y][d9]__[^T][14]__[M-^][de]__[M-^O][8f]__[>][3e]__[^T][14]_
Orig Key:_[X][58]__[R][52]__[F][46]__[m][6d]__[B][42]__[r][72]_
CT 4 verify:_[M-Y][d9]__[^T][14]__[M-^][de]__[M-^O][8f]__[>][3e]__[^T][14]
   _
We have Verification also.
Limits: nednodes=4:ppn=8,nodes=4:ppn=8,walltime=04:00:00
Resources:     cput=10:45:06,mem=1442340kb,vmem=8479256kb,walltime=00:20:19

45697770
Lead at t=1 core=1, m=0, key1stB_I_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[M-E][c5]__[M-m][ed]__[M-t][f4]__[M-l][ec]__[M-%][a5]__[^@][00]_
Recovered Key(EP-1):_[I][49]__[b][62]__[N][4e]__[v][76]__[1][31]__[E][45]_
Chain SP:_[I][49]__[b][62]__[N][4e]__[v][76]__[1][31]__[E][45]_
```

```
Orig CT:_[F][46]__[^O][0f]__[x][78]__[b][62]__[M-^\][9c]__[s][73]_
Orig Key:_[I][49]__[b][62]__[N][4e]__[v][76]__[1][31]__[E][45]_
CT 4 verify:_[F][46]__[^O][0f]__[x][78]__[b][62]__[M-^\][9c]__[s][73]_
We have Verification also.
Limits: nednodes=4:ppn=8,nodes=4:ppn=8,walltime=04:00:00
Resources:     cput=10:44:39,mem=1443232kb,vmem=8480248kb,walltime=00:20:17

45697769
Lead at t=1 core=1, m=0, key1stB_t_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[M-)][a9]__[M-N][ce]__[M-E][c5]__[0][30]__[K][4b]__[4][34]_
Recovered Key(EP-1):_[t][74]__[L][4c]__[L][4c]__[J][4a]__[o][6f]__[1][31]_
Chain SP:_[t][74]__[L][4c]__[L][4c]__[J][4a]__[o][6f]__[1][31]_
Orig CT:_[M-,][ac]__[P][50]__[M-4][b4]__[/][2f]__[_][5f]__[M-b][e2]_
Orig Key:_[t][74]__[L][4c]__[L][4c]__[J][4a]__[o][6f]__[1][31]_
CT 4 verify:_[M-,][ac]__[P][50]__[M-4][b4]__[/][2f]__[_][5f]__[M-b][e2]_
We have Verification also.
Limits: nednodes=4:ppn=8,nodes=4:ppn=8,walltime=04:00:00
Resources:     cput=10:43:35,mem=1445372kb,vmem=8481520kb,walltime=00:20:16

45697768
Lead at t=1 core=1, m=0, key1stB_j_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[M-^][de]__[^H][08]__[L][4c]__[M-U][d5]__[Y][59]__[A][41]_
Recovered Key(EP-1):_[j][6a]__[a][61]__[m][6d]__[A][41]__[i][69]__[D][44]_
Chain SP:_[j][6a]__[a][61]__[m][6d]__[A][41]__[i][69]__[D][44]_
Orig CT:_[A][41]__[^U][15]__[z][7a]__[M-^C][83]__[x][78]__[^]][1d]_
Orig Key:_[j][6a]__[a][61]__[m][6d]__[A][41]__[i][69]__[D][44]_
CT 4 verify:_[A][41]__[^U][15]__[z][7a]__[M-^C][83]__[x][78]__[^]][1d]_
We have Verification also.
Limits: nednodes=4:ppn=8,nodes=4:ppn=8,walltime=04:00:00
Resources:     cput=10:44:53,mem=1435164kb,vmem=8481364kb,walltime=00:20:20

45697747
][0a]__[M-o][ef]__[f][66]__[M-^M][8d]__[@][40]_
Recovered Key(EP-1):_[J][4a]__[I][49]__[r][72]__[3][33]__[G][47]__[9][39]_
Chain SP:_[J][4a]__[I][49]__[r][72]__[3][33]__[G][47]__[9][39]_
Orig CT:_[M-_][df]__[M-^M][8d]__[M-.][ae]__[M-b][e2]__[Z][5a]__[M-K][cb]_
```

```
Orig Key:_[J][4a]__[I][49]__[r][72]__[3][33]__[G][47]__[9][39]_
CT 4 verify:_[M-_][df]__[M-^M][8d]__[M-.][ae]__[M-b][e2]__[Z][5a]__[M-K][
   cb]_
We have Verification also.
Limits: nednodes=4:ppn=8,nodes=4:ppn=8,walltime=03:00:00
Resources:     cput=10:44:31,mem=1445880kb,vmem=8480920kb,walltime=00:20:19

45697746
Lead at t=1 core=1, m=0, key1stB_U_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[M-m][ed]__[X][58]__[\][5c]__[M-5][b5]__[E][45]__[^N][0e]_
Recovered Key(EP-1):_[U][55]__[3][33]__[Y][59]__[A][41]__[e][65]__[0][30]_
Chain SP:_[U][55]__[3][33]__[Y][59]__[A][41]__[e][65]__[0][30]_
Orig CT:_[M-^F][86]__[M-#][a3]__[N][4e]__[M-&][a6]__[^@][00]__[M-@][c0]_
Orig Key:_[U][55]__[3][33]__[Y][59]__[A][41]__[e][65]__[0][30]_
CT 4 verify:_[M-^F][86]__[M-#][a3]__[N][4e]__[M-&][a6]__[^@][00]__[M-@][c0
   ]_
We have Verification also.
Limits: nednodes=4:ppn=8,nodes=4:ppn=8,walltime=03:00:00
Resources:     cput=10:43:31,mem=1442520kb,vmem=8480284kb,walltime=00:20:16

45697745
Lead at t=1 core=1, m=0, key1stB_2_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[>][3e]__[-][2d]__[^Z][1a]__[M-,][ac]__[+][2b]__[M-^Q][91]_
Recovered Key(EP-1):_[2][32]__[6][36]__[k][6b]__[A][41]__[S][53]__[D][44]_
Chain SP:_[2][32]__[6][36]__[k][6b]__[A][41]__[S][53]__[D][44]_
Orig CT:_[M-6][b6]__[b][62]__[M-9][b9]__[M-^X][98]__[9][39]__[M-l][ec]_
Orig Key:_[2][32]__[6][36]__[k][6b]__[A][41]__[S][53]__[D][44]_
CT 4 verify:_[M-6][b6]__[b][62]__[M-9][b9]__[M-^X][98]__[9][39]__[M-l][ec]
   _
We have Verification also.
Limits: nednodes=4:ppn=8,nodes=4:ppn=8,walltime=03:00:00
Resources:     cput=10:44:42,mem=1440068kb,vmem=8478928kb,walltime=00:20:17

45697744
Lead at t=1 core=1, m=0, key1stB_p_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
```

```
Found CT:_[^][5e]__[M-^V][96]__[7][37]__[M-[][db]__[M-u][f5]__[M-^O][8f]_
Recovered Key(EP-1):_[p][70]__[p][70]__[2][32]__[3][33]__[P][50]__[C][43]_
Chain SP:_[p][70]__[p][70]__[2][32]__[3][33]__[P][50]__[C][43]_
Orig CT:_[N][4e]__[I][49]__[?][3f]__[M-^F][86]__[M-^@][80]__[^B][02]_
Orig Key:_[p][70]__[p][70]__[2][32]__[3][33]__[P][50]__[C][43]_
CT 4 verify:_[N][4e]__[I][49]__[?][3f]__[M-^F][86]__[M-^@][80]__[^B][02]_
We have Verification also.
Limits: nednodes=4:ppn=8,nodes=4:ppn=8,walltime=03:00:00
Resources:     cput=10:43:40,mem=1446656kb,vmem=8480468kb,walltime=00:20:18

45697743
Lead at t=1 core=1, m=0, key1stB_7_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[M-w][f7]__[O][4f]__[s][73]__[l][6c]__[M-{][fb]__[l][6c]_
Recovered Key(EP-1):_[7][37]__[X][58]__[k][6b]__[Z][5a]__[M][4d]__[f][66]_
Chain SP:_[7][37]__[X][58]__[k][6b]__[Z][5a]__[M][4d]__[f][66]_
Orig CT:_[M-^I][89]__[j][6a]__[^C][03]__[M-<][bc]__[M-Y][d9]__[T][54]_
Orig Key:_[7][37]__[X][58]__[k][6b]__[Z][5a]__[M][4d]__[f][66]_
CT 4 verify:_[M-^I][89]__[j][6a]__[^C][03]__[M-<][bc]__[M-Y][d9]__[T][54]_
We have Verification also.
Limits: nednodes=4:ppn=8,nodes=4:ppn=8,walltime=03:00:00
Resources:     cput=10:44:40,mem=1443360kb,vmem=8481156kb,walltime=00:20:19

45697742
Lead at t=1 core=1, m=0, key1stB_O_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[u][75]__[%][25]__[^E][05]__[^F][06]__[M-^C][83]__[M-A][c1]_
Recovered Key(EP-1):_[O][4f]__[Y][59]__[R][52]__[w][77]__[1][31]__[r][72]_
Chain SP:_[O][4f]__[Y][59]__[R][52]__[w][77]__[1][31]__[r][72]_
Orig CT:_[^_][1f]__[M-^@][80]__[M-X][d8]__[?][3f]__[M-^U][95]__[M-R][d2]_
Orig Key:_[O][4f]__[Y][59]__[R][52]__[w][77]__[1][31]__[r][72]_
CT 4 verify:_[^_][1f]__[M-^@][80]__[M-X][d8]__[?][3f]__[M-^U][95]__[M-R][
   d2]_
We have Verification also.
Limits: nednodes=4:ppn=8,nodes=4:ppn=8,walltime=03:00:00
Resources:     cput=10:45:36,mem=1440764kb,vmem=8480248kb,walltime=00:20:18

45697741
```

```
Lead at t=1 core=1, m=0, key1stB_8_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[}][7d]__[M-^M][8d]__[M-+][ab]__[~][7e]__[^X][18]__[M--][ad]_
Recovered Key(EP-1):_[8][38]__[W][57]__[F][46]__[r][72]__[m][6d]__[1][31]_
Chain SP:_[8][38]__[W][57]__[F][46]__[r][72]__[m][6d]__[1][31]_
Orig CT:_[!][21]__[w][77]__[M-7][b7]__[M-‘][e0]__[p][70]__[M-e][e5]_
Orig Key:_[8][38]__[W][57]__[F][46]__[r][72]__[m][6d]__[1][31]_
CT 4 verify:_[!][21]__[w][77]__[M-7][b7]__[M-‘][e0]__[p][70]__[M-e][e5]_
We have Verification also.
Limits: nednodes=4:ppn=8,nodes=4:ppn=8,walltime=03:00:00
Resources:     cput=10:43:58,mem=1441208kb,vmem=8480144kb,walltime=00:20:18

45697739
Lead at t=1 core=1, m=0, key1stB_z_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[M-^K][8b]__[M-^A][81]__[M-^_][9f]__[%][25]__[M-d][e4]__[M-^\][9
   c]_
Recovered Key(EP-1):_[z][7a]__[W][57]__[U][55]__[I][49]__[m][6d]__[F][46]_
Chain SP:_[z][7a]__[W][57]__[U][55]__[I][49]__[m][6d]__[F][46]_
Orig CT:_[^E][05]__[M-^J][8a]__[M-:][ba]__[M-^X][98]__[M-f][e6]__[M-e][e5]
   _
Orig Key:_[z][7a]__[W][57]__[U][55]__[I][49]__[m][6d]__[F][46]_
CT 4 verify:_[^E][05]__[M-^J][8a]__[M-:][ba]__[M-^X][98]__[M-f][e6]__[M-e
   ][e5]_
We have Verification also.
Limits: nednodes=4:ppn=8,nodes=4:ppn=8,walltime=03:00:00
Resources:     cput=10:44:45,mem=1444280kb,vmem=8482100kb,walltime=00:20:20

45697740
Lead at t=1 core=1, m=0, key1stB_k_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[M-^D][84]__[M-A][c1]__[M-^Y][99]__[M-‘][e0]__[M-s][f3]__[W][57]
   _
Recovered Key(EP-1):_[k][6b]__[j][6a]__[4][34]__[v][76]__[U][55]__[g][67]_
Chain SP:_[k][6b]__[j][6a]__[4][34]__[v][76]__[U][55]__[g][67]_
Orig CT:_[]][5d]__[M-.][ae]__[M-=][bd]__[M-k][eb]__[O][4f]__[M-^E][85]_
Orig Key:_[k][6b]__[j][6a]__[4][34]__[v][76]__[U][55]__[g][67]_
CT 4 verify:_[]][5d]__[M-.][ae]__[M-=][bd]__[M-k][eb]__[O][4f]__[M-^E][85]
```

```
   _
We have Verification also.
Limits: neednodes=4:ppn=8,nodes=4:ppn=8,walltime=03:00:00
Resources:     cput=10:43:46,mem=1442380kb,vmem=8480468kb,walltime=00:20:20

45697738
Lead at t=1 core=1, m=0, key1stB_s_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[M-[][db]__[M-=][bd]__[M-^@][80]__[^Q][11]__[r][72]__[G][47]_
Recovered Key(EP-1):_[s][73]__[0][30]__[C][43]__[m][6d]__[D][44]__[a][61]_
Chain SP:_[s][73]__[0][30]__[C][43]__[m][6d]__[D][44]__[a][61]_
Orig CT:_[M-/][af]__[M-^F][86]__[2][32]__[M-^S][93]__[M-^X][98]__[M-f][e6]
   _
Orig Key:_[s][73]__[0][30]__[C][43]__[m][6d]__[D][44]__[a][61]_
CT 4 verify:_[M-/][af]__[M-^F][86]__[2][32]__[M-^S][93]__[M-^X][98]__[M-f
   ][e6]_
We have Verification also.
Limits: neednodes=4:ppn=8,nodes=4:ppn=8,walltime=03:00:00
Resources:     cput=10:43:36,mem=1446768kb,vmem=8480472kb,walltime=00:20:18

45697717
Lead at t=1 core=1, m=0, key1stB_U_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[M-<][bc]__[M--][ad]__[M-^P][90]__[O][4f]__[M-i][e9]__[M-_][df]_
Recovered Key(EP-1):_[U][55]__[T][54]__[v][76]__[D][44]__[v][76]__[q][71]_
Chain SP:_[U][55]__[T][54]__[v][76]__[D][44]__[v][76]__[q][71]_
Orig CT:_[7][37]__[t][74]__[^[][1b]__[M- ][a0]__[M-^E][85]__[U][55]_
Orig Key:_[U][55]__[T][54]__[v][76]__[D][44]__[v][76]__[q][71]_
CT 4 verify:_[7][37]__[t][74]__[^[][1b]__[M- ][a0]__[M-^E][85]__[U][55]_
We have Verification also.
Limits: neednodes=4:ppn=8,nodes=4:ppn=8,walltime=02:00:00
Resources:     cput=10:45:42,mem=1439728kb,vmem=8480116kb,walltime=00:20:20

45697715
Lead at t=1 core=1, m=0, key1stB_I_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[M-)][a9]__[M-E][c5]__[M-,][ac]__[g][67]__[z][7a]__[M-L][cc]_
Recovered Key(EP-1):_[I][49]__[U][55]__[8][38]__[7][37]__[d][64]__[G][47]_
```

```
Chain SP:_[I][49]__[U][55]__[8][38]__[7][37]__[d][64]__[G][47]_
Orig CT:_[o][6f]__[o][6f]__[M-^L][8c]__[L][4c]__[M-W][d7]__[M-^[][9b]_
Orig Key:_[I][49]__[U][55]__[8][38]__[7][37]__[d][64]__[G][47]_
CT 4 verify:_[o][6f]__[o][6f]__[M-^L][8c]__[L][4c]__[M-W][d7]__[M-^[][9b]_
We have Verification also.
Limits: nednodes=4:ppn=8,nodes=4:ppn=8,walltime=02:00:00
Resources:     cput=10:43:47,mem=1435576kb,vmem=8480404kb,walltime=00:20:19

45697716
Lead at t=1 core=1, m=0, key1stB_t_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[#][23]__[M-^Q][91]__[4][34]__[O][4f]__[M-^?][ff]__[M-d][e4]_
Recovered Key(EP-1):_[t][74]__[P][50]__[7][37]__[3][33]__[A][41]__[3][33]_
Chain SP:_[t][74]__[P][50]__[7][37]__[3][33]__[A][41]__[3][33]_
Orig CT:_[M-.][ae]__[M-/][af]__[M-t][f4]__[E][45]__[T][54]__[^]][1d]_
Orig Key:_[t][74]__[P][50]__[7][37]__[3][33]__[A][41]__[3][33]_
CT 4 verify:_[M-.][ae]__[M-/][af]__[M-t][f4]__[E][45]__[T][54]__[^]][1d]_
We have Verification also.
Limits: nednodes=4:ppn=8,nodes=4:ppn=8,walltime=02:00:00
Resources:     cput=10:43:42,mem=1444636kb,vmem=8480272kb,walltime=00:20:18

45697714
Lead at t=1 core=1, m=0, key1stB_i_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[M-<][bc]__[M-^G][87]__[M-P][d0]__[M-#][a3]__[Z][5a]__[#][23]_
Recovered Key(EP-1):_[i][69]__[h][68]__[H][48]__[6][36]__[Z][5a]__[T][54]_
Chain SP:_[i][69]__[h][68]__[H][48]__[6][36]__[Z][5a]__[T][54]_
Orig CT:_[R][52]__[M-Z][da]__[M-5][b5]__[_][5f]__[M-^E][85]__[}][7d]_
Orig Key:_[i][69]__[h][68]__[H][48]__[6][36]__[Z][5a]__[T][54]_
CT 4 verify:_[R][52]__[M-Z][da]__[M-5][b5]__[_][5f]__[M-^E][85]__[}][7d]_
We have Verification also.
Limits: nednodes=4:ppn=8,nodes=4:ppn=8,walltime=02:00:00
Resources:     cput=10:45:02,mem=1439196kb,vmem=8478860kb,walltime=00:20:17

45697713
Lead at t=1 core=1, m=0, key1stB_J_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[M-I][c9]__[,][2c]__[M-~][fe]__[-][2d]__[^X][18]__[M-^G][87]_
```

```
Recovered Key(EP-1):_[J][4a]__[2][32]__[D][44]__[S][53]__[M][4d]__[T][54]_
Chain SP:_[J][4a]__[2][32]__[D][44]__[S][53]__[M][4d]__[T][54]_
Orig CT:_[M-p][f0]__[^E][05]__[^_][1f]__[Y][59]__[^E][05]__[M-"][a2]_
Orig Key:_[J][4a]__[2][32]__[D][44]__[S][53]__[M][4d]__[T][54]_
CT 4 verify:_[M-p][f0]__[^E][05]__[^_][1f]__[Y][59]__[^E][05]__[M-"][a2]_
We have Verification also.
Limits: nednodes=4:ppn=8,nodes=4:ppn=8,walltime=02:00:00
Resources:     cput=10:43:38,mem=1438476kb,vmem=8480280kb,walltime=00:20:18

45697712
Lead at t=1 core=1, m=0, key1stB_d_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[M-^Z][9a]__[s][73]__[M-z][fa]__[i][69]__[‘][60]__[e][65]_
Recovered Key(EP-1):_[d][64]__[1][31]__[j][6a]__[f][66]__[p][70]__[P][50]_
Chain SP:_[d][64]__[1][31]__[j][6a]__[f][66]__[p][70]__[P][50]_
Orig CT:_[x][78]__[M-^V][96]__[1][31]__[’][27]__[M-o][ef]__[2][32]_
Orig Key:_[d][64]__[1][31]__[j][6a]__[f][66]__[p][70]__[P][50]_
CT 4 verify:_[x][78]__[M-^V][96]__[1][31]__[’][27]__[M-o][ef]__[2][32]_
We have Verification also.
Limits: nednodes=4:ppn=8,nodes=4:ppn=8,walltime=02:00:00
Resources:     cput=10:44:31,mem=1443028kb,vmem=8481552kb,walltime=00:20:17

45697711
Lead at t=1 core=1, m=0, key1stB_c_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[^O][0f]__[M-^@][80]__[l][6c]__[T][54]__[M-:][ba]__[M-^D][84]_
Recovered Key(EP-1):_[c][63]__[K][4b]__[j][6a]__[W][57]__[3][33]__[d][64]_
Chain SP:_[c][63]__[K][4b]__[j][6a]__[W][57]__[3][33]__[d][64]_
Orig CT:_[^@][00]__[M-U][d5]__[^[][1b]__[\][5c]__[6][36]__[M-A][c1]_
Orig Key:_[c][63]__[K][4b]__[j][6a]__[W][57]__[3][33]__[d][64]_
CT 4 verify:_[^@][00]__[M-U][d5]__[^[][1b]__[\][5c]__[6][36]__[M-A][c1]_
We have Verification also.
Limits: nednodes=4:ppn=8,nodes=4:ppn=8,walltime=02:00:00
Resources:     cput=10:43:45,mem=1446468kb,vmem=8480276kb,walltime=00:20:16

45697710
Lead at t=1 core=1, m=0, key1stB_s_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
```

```
Found CT:_[&][26]__[M-=][bd]__[M-^Y][99]__[^R][12]__[P][50]__[J][4a]_
Recovered Key(EP-1):_[s][73]__[F][46]__[B][42]__[x][78]__[N][4e]__[g][67]_
Chain SP:_[s][73]__[F][46]__[B][42]__[x][78]__[N][4e]__[g][67]_
Orig CT:_[M-\][dc]__[M-(][a8]__[+][2b]__[^^][1e]__[t][74]__[M-g][e7]_
Orig Key:_[s][73]__[F][46]__[B][42]__[x][78]__[N][4e]__[g][67]_
CT 4 verify:_[M-\][dc]__[M-(][a8]__[+][2b]__[^^][1e]__[t][74]__[M-g][e7]_
We have Verification also.
Limits: nednodes=4:ppn=8,nodes=4:ppn=8,walltime=02:00:00
Resources:     cput=10:43:40,mem=1442372kb,vmem=8480468kb,walltime=00:20:19

45697709
Lead at t=1 core=1, m=0, key1stB_v_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[s][73]__[l][6c]__[^^][1e]__[M-8][b8]__[9][39]__[)][29]_
Recovered Key(EP-1):_[v][76]__[d][64]__[O][4f]__[U][55]__[F][46]__[V][56]_
Chain SP:_[v][76]__[d][64]__[O][4f]__[U][55]__[F][46]__[V][56]_
Orig CT:_[M-^O][8f]__[M-[][db]__[j][6a]__[^T][14]__[M-b][e2]__[M-)][a9]_
Orig Key:_[v][76]__[d][64]__[O][4f]__[U][55]__[F][46]__[V][56]_
CT 4 verify:_[M-^O][8f]__[M-[][db]__[j][6a]__[^T][14]__[M-b][e2]__[M-)][a9
   ]_
We have Verification also.
Limits: nednodes=4:ppn=8,nodes=4:ppn=8,walltime=02:00:00
Resources:     cput=10:43:31,mem=1444296kb,vmem=8480404kb,walltime=00:20:16

45697708
Lead at t=1 core=1, m=0, key1stB_Z_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[M-F][c6]__[M-B][c2]__[M-l][ec]__[.][2e]__[^Z][1a]__[M-w][f7]_
Recovered Key(EP-1):_[Z][5a]__[A][41]__[R][52]__[R][52]__[M][4d]__[3][33]_
Chain SP:_[Z][5a]__[A][41]__[R][52]__[R][52]__[M][4d]__[3][33]_
Orig CT:_[^C][03]__[^L][0c]__[O][4f]__[O][4f]__[V][56]__[^H][08]_
Orig Key:_[Z][5a]__[A][41]__[R][52]__[R][52]__[M][4d]__[3][33]_
CT 4 verify:_[^C][03]__[^L][0c]__[O][4f]__[O][4f]__[V][56]__[^H][08]_
We have Verification also.
Limits: nednodes=4:ppn=8,nodes=4:ppn=8,walltime=02:00:00
Resources:     cput=10:43:33,mem=1447660kb,vmem=8480468kb,walltime=00:20:19

45697686
```

```
Lead at t=1 core=1, m=0, key1stB_d_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[M-p][f0]__[ ][20]__[M-j][ea]__[x][78]__[M-B][c2]__[M- ][a0]_
Recovered Key(EP-1):_[d][64]__[7][37]__[1][31]__[E][45]__[L][4c]__[Z][5a]_
Chain SP:_[d][64]__[7][37]__[1][31]__[E][45]__[L][4c]__[Z][5a]_
Orig CT:_[M-5][b5]__[M-[][db]__[M-^K][8b]__[M-^K][8b]__[i][69]__[M-j][ea]_
Orig Key:_[d][64]__[7][37]__[1][31]__[E][45]__[L][4c]__[Z][5a]_
CT 4 verify:_[M-5][b5]__[M-[][db]__[M-^K][8b]__[M-^K][8b]__[i][69]__[M-j][
   ea]_
We have Verification also.
Limits: nednodes=4:ppn=8,nodes=4:ppn=8,walltime=01:00:00
Resources:     cput=10:43:53,mem=1444184kb,vmem=8480404kb,walltime=00:20:18

45697687
Lead at t=1 core=1, m=0, key1stB_f_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[$][24]__[y][79]__[M-^_][9f]__[M-2][b2]__[f][66]__[M][4d]_
Recovered Key(EP-1):_[f][66]__[H][48]__[r][72]__[D][44]__[2][32]__[p][70]_
Chain SP:_[f][66]__[H][48]__[r][72]__[D][44]__[2][32]__[p][70]_
Orig CT:_[M-3][b3]__[M-^B][82]__[(][28]__[M-^V][96]__[M-^Z][9a]__[M-^B
   ][82]_
Orig Key:_[f][66]__[H][48]__[r][72]__[D][44]__[2][32]__[p][70]_
CT 4 verify:_[M-3][b3]__[M-^B][82]__[(][28]__[M-^V][96]__[M-^Z][9a]__[M-^B
   ][82]_
We have Verification also.
Limits: nednodes=4:ppn=8,nodes=4:ppn=8,walltime=01:00:00
Resources:     cput=10:45:03,mem=1443296kb,vmem=8480384kb,walltime=00:20:18

45697685
Lead at t=1 core=1, m=0, key1stB_3_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[M][4d]__[L][4c]__[>][3e]__[M-l][ec]__[j][6a]__[M][4d]_
Recovered Key(EP-1):_[3][33]__[e][65]__[W][57]__[6][36]__[D][44]__[A][41]_
Chain SP:_[3][33]__[e][65]__[W][57]__[6][36]__[D][44]__[A][41]_
Orig CT:_[M-@][c0]__[^O][0f]__[M-t][f4]__[M-_][df]__[M-M][cd]__[*][2a]_
Orig Key:_[3][33]__[e][65]__[W][57]__[6][36]__[D][44]__[A][41]_
CT 4 verify:_[M-@][c0]__[^O][0f]__[M-t][f4]__[M-_][df]__[M-M][cd]__[*][2a]
   _
```

```
We have Verification also.
Limits: neednodes=4:ppn=8,nodes=4:ppn=8,walltime=01:00:00
Resources:     cput=10:45:05,mem=1445036kb,vmem=8481264kb,walltime=00:20:19

45697684
Lead at t=1 core=1, m=0, key1stB_X_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[V][56]__[^?][7f]__[T][54]__[M-p][f0]__[M-^C][83]__[v][76]_
Recovered Key(EP-1):_[X][58]__[4][34]__[Y][59]__[B][42]__[f][66]__[X][58]_
Chain SP:_[X][58]__[4][34]__[Y][59]__[B][42]__[f][66]__[X][58]_
Orig CT:_[a][61]__[M-^E][85]__[2][32]__[M-G][c7]__[M-^C][83]__[^W][17]_
Orig Key:_[X][58]__[4][34]__[Y][59]__[B][42]__[f][66]__[X][58]_
CT 4 verify:_[a][61]__[M-^E][85]__[2][32]__[M-G][c7]__[M-^C][83]__[^W][17]
   _
We have Verification also.
Limits: neednodes=4:ppn=8,nodes=4:ppn=8,walltime=01:00:00
Resources:     cput=10:43:25,mem=1440876kb,vmem=8480688kb,walltime=00:20:15

45697683
Lead at t=1 core=1, m=0, key1stB_J_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[M-7][b7]__[M-^K][8b]__[M-^G][87]__[M-I][c9]__[M-2][b2]__[^L][0c
   ]_
Recovered Key(EP-1):_[J][4a]__[d][64]__[W][57]__[d][64]__[g][67]__[E][45]_
Chain SP:_[J][4a]__[d][64]__[W][57]__[d][64]__[g][67]__[E][45]_
Orig CT:_[M-x][f8]__[M-%][a5]__[^T][14]__[*][2a]__[M-5][b5]__[$][24]_
Orig Key:_[J][4a]__[d][64]__[W][57]__[d][64]__[g][67]__[E][45]_
CT 4 verify:_[M-x][f8]__[M-%][a5]__[^T][14]__[*][2a]__[M-5][b5]__[$][24]_
We have Verification also.
Limits: neednodes=4:ppn=8,nodes=4:ppn=8,walltime=01:00:00
Resources:     cput=10:45:08,mem=1442972kb,vmem=8480308kb,walltime=00:20:19

45697682
Lead at t=1 core=1, m=0, key1stB_t_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[M-$][a4]__[9][39]__[M-^\][9c]__[M-e][e5]__[^C][03]__[s][73]_
Recovered Key(EP-1):_[t][74]__[Q][51]__[N][4e]__[k][6b]__[N][4e]__[8][38]_
Chain SP:_[t][74]__[Q][51]__[N][4e]__[k][6b]__[N][4e]__[8][38]_
```

```
Orig CT:_[k][6b]__[h][68]__[M-C][c3]__[M-q][f1]__[M-D][c4]__[M-v][f6]_
Orig Key:_[t][74]__[Q][51]__[N][4e]__[k][6b]__[N][4e]__[8][38]_
CT 4 verify:_[k][6b]__[h][68]__[M-C][c3]__[M-q][f1]__[M-D][c4]__[M-v][f6]_
We have Verification also.
Limits: nednodes=4:ppn=8,nodes=4:ppn=8,walltime=01:00:00
Resources:     cput=10:43:38,mem=1443940kb,vmem=8479868kb,walltime=00:20:16

45697680
Lead at t=1 core=1, m=0, key1stB_0_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[M-^J][8a]__[M-i][e9]__[M-S][d3]__[E][45]__[ ][20]__[Q][51]_
Recovered Key(EP-1):_[0][30]__[r][72]__[n][6e]__[o][6f]__[9][39]__[g][67]_
Chain SP:_[0][30]__[r][72]__[n][6e]__[o][6f]__[9][39]__[g][67]_
Orig CT:_[M-^D][84]__[M-^B][82]__[{][7b]__[&][26]__[^Q][11]__[M-‘][e0]_
Orig Key:_[0][30]__[r][72]__[n][6e]__[o][6f]__[9][39]__[g][67]_
CT 4 verify:_[M-^D][84]__[M-^B][82]__[{][7b]__[&][26]__[^Q][11]__[M-‘][e0]
   _
We have Verification also.
Limits: nednodes=4:ppn=8,nodes=4:ppn=8,walltime=01:00:00
Resources:     cput=10:44:49,mem=1447104kb,vmem=8482356kb,walltime=00:20:19

45697681
Lead at t=1 core=1, m=0, key1stB_y_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[M-W][d7]__[M-p][f0]__[O][4f]__[J][4a]__[M-^Y][99]__[M-5][b5]_
Recovered Key(EP-1):_[y][79]__[F][46]__[3][33]__[w][77]__[6][36]__[M][4d]_
Chain SP:_[y][79]__[F][46]__[3][33]__[w][77]__[6][36]__[M][4d]_
Orig CT:_[M-o][ef]__[H][48]__[ ][09]__[:][3a]__[M-c][e3]__[%][25]_
Orig Key:_[y][79]__[F][46]__[3][33]__[w][77]__[6][36]__[M][4d]_
CT 4 verify:_[M-o][ef]__[H][48]__[  ][09]__[:][3a]__[M-c][e3]__[%][25]_
We have Verification also.
Limits: nednodes=4:ppn=8,nodes=4:ppn=8,walltime=01:00:00
Resources:     cput=10:43:42,mem=1444744kb,vmem=8480288kb,walltime=00:20:19

45697679
Lead at t=1 core=1, m=0, key1stB_s_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[M-^E][85]__[M-n][ee]__[h][68]__[O][4f]__[M-x][f8]__[M-4][b4]_
```

```
Recovered Key(EP-1):_[s][73]__[z][7a]__[j][6a]__[c][63]__[V][56]__[2][32]_
Chain SP:_[s][73]__[z][7a]__[j][6a]__[c][63]__[V][56]__[2][32]_
Orig CT:_[M-d][e4]__[M-^@][80]__[M-_][df]__[^N][0e]__[M-9][b9]__[f][66]_
Orig Key:_[s][73]__[z][7a]__[j][6a]__[c][63]__[V][56]__[2][32]_
CT 4 verify:_[M-d][e4]__[M-^@][80]__[M-_][df]__[^N][0e]__[M-9][b9]__[f
   ][66]_
We have Verification also.
Limits: nednodes=4:ppn=8,nodes=4:ppn=8,walltime=01:00:00
Resources:     cput=10:43:34,mem=1443928kb,vmem=8480276kb,walltime=00:20:20

45697678
Lead at t=1 core=1, m=0, key1stB_3_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[M-"][a2]__[b][62]__[^M][0d]__[M-‘][e0]__[M-^Z][9a]__[1][31]_
Recovered Key(EP-1):_[3][33]__[q][71]__[C][43]__[L][4c]__[8][38]__[W][57]_
Chain SP:_[3][33]__[q][71]__[C][43]__[L][4c]__[8][38]__[W][57]_
Orig CT:_[L][4c]__[9][39]__[_][5f]__[M-O][cf]__[M-c][e3]__[8][38]_
Orig Key:_[3][33]__[q][71]__[C][43]__[L][4c]__[8][38]__[W][57]_
CT 4 verify:_[L][4c]__[9][39]__[_][5f]__[M-O][cf]__[M-c][e3]__[8][38]_
We have Verification also.
Limits: nednodes=4:ppn=8,nodes=4:ppn=8,walltime=01:00:00
Resources:     cput=10:43:25,mem=1447720kb,vmem=8480404kb,walltime=00:20:17
```

A.2 Pushing boundaries tactics or probing strategy

A.2.1 Pushing T

x=T coverage, y=results

```
IncT1048576t1024c2145386496m4hrs_bestCvrg7.o45881810
IncT1048576t1024c2145386496m4hrs_bestCvrg7.e45881810
IncT4096t1024c2145386496m4hrs_bestCvrg0.o45881779
IncT4096t1024c2145386496m4hrs_bestCvrg0.o45881777
IncT4096t1024c2145386496m4hrs_bestCvrg0.e45881777
IncT4096t1024c2145386496m4hrs_bestCvrg0.e45881779
IncT4096t1024c2145386496m4hrs_bestCvrg0.e45881775
IncT4096t1024c2145386496m4hrs_bestCvrg0.o45881775
IncT4096t1024c2145386496m4hrs_bestCvrg0.e45881769
IncT4096t1024c2145386496m4hrs_bestCvrg0.o45881769
IncT4096t1024c1072693248m4hrs_bestCvrg0.e45881767
IncT4096t1024c1072693248m4hrs_bestCvrg0.o45881767
IncT4096t1024c1072693248m4hrs_bestCvrg0.o45881765
IncT4096t1024c1072693248m4hrs_bestCvrg0.e45881765
IncT4096t1024c1072693248m4hrs_bestCvrg0.o45881764
IncT4096t1024c1072693248m4hrs_bestCvrg0.o45881763
IncT4096t1024c1072693248m4hrs_bestCvrg0.e45881763
IncT4096t1024c1072693248m4hrs_bestCvrg0.e45881764
IncT4096t1024c1072693248m4hrs_bestCvrg0.e45881762
IncT4096t1024c1072693248m4hrs_bestCvrg0.o45881762
IncT4096t1024c1072693248m4hrs_bestCvrg0.o45881761
IncT4096t1024c1072693248m4hrs_bestCvrg0.o45881758
IncT4096t1024c1072693248m4hrs_bestCvrg0.e45881761
IncT4096t1024c1072693248m4hrs_bestCvrg0.e45881758
IncT4096t1024c1072693248m4hrs_bestCvrg0.e45881760
IncT4096t1024c1072693248m4hrs_bestCvrg0.o45881760
IncT4096t1024c1072693248m4hrs_bestCvrg0.e45881757
IncT4096t1024c1072693248m4hrs_bestCvrg0.o45881757
```

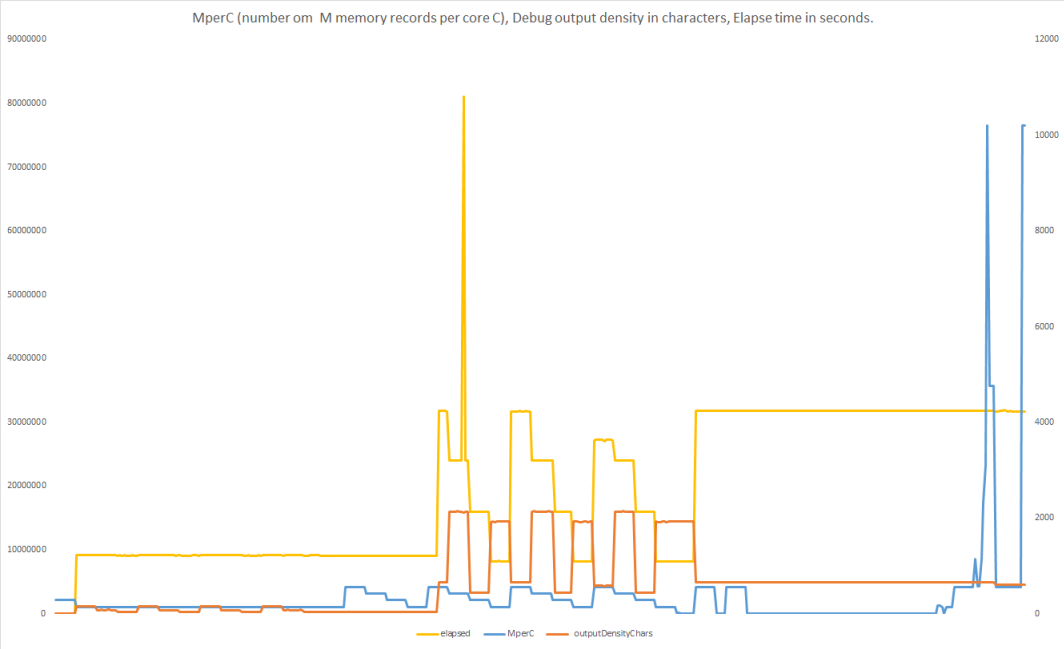

Figure A.1: Relating density of output to job elapsed time in seconds

A.2.2 Decreasing output density

The output to file, like network and other IO, is handled by MPI. It was important to execute runs that are not output dense and compare the results.

Figure A.1 Plot cross reference between MperC Char/Byte Density of output and elapsed time in seconds.

```
Job ID: 45941096 45941097 45941098 45941099 45941100 45941101 45941102
   45941103 45941104 45941105 45941106 45941107 45941109 45941110 45941111
    45941112 45941113 45941114 45941115 45941116
The number of booked cores is 1024 (1 Parent and 1023 children)
plaintext is: PTPTPT
Random key: (multiple - details below) and it is used as the start point
   of the first chain.
After encrypting the plaintext T times (4096) the chain end point becomes
   : (multiple - details below)
Ciphertext Challenge: (multiple - details below)
CT decrypts back to: (multiple - details below)
```

```
output log: (multiple)
Output size (printf overhead): around 12400 lines 940000 characters.
error log: (multiple)
Job log: (multiple)
source code file:(multiple)
job submission script:
object file:(multiple)
Cores: 1024
usableCores:1023
PID:(multiple)
Date: (multiple)
M=1072693248
C=1024
W=04:00:00
T=4096
B=48
E=2
H=0
L=0
S=1
R=GPC
notes=(multiple)
```

Observation: Some jobs were successful and some were not. Regardless of how much wall-time was asked, the jobs that had T=4096, C=1024, and M=1,072,693,248 finished in around 01:16:20 hours.

Some jobs that had T=4096, C=1024, and M=2,145,386,496 Stopped after about 04:00:00 hours as the GPC killed the jobs for going over the time. This can be justified as M is twice the size. Looking at the output file (Ex: $LowD4096t1024c2145386496m4hrs_bestCvrg0.o45941114$) the time expires while the child processes are generating the starting point. This sample will be used to decrease the the test density further more and remeasure results.

Some jobs that had T=4096, C=1024, and M=2,145,386,496 Stopped after about 02:31:00 hours. It is not clear why they stopped but they stop around the same place as the other failed jobs. That is, after generating 2097152 LFSR for starting points.

Next action: Decrease output density further more. With 1024 cores we are allowed 16 hours. Queue jobs for M=1,072,693,248 under these condition.

```
LowD4096t1024c1072693248m4hrs_bestCvrg0.o45941096
45941096
Lead at t=1 core=1, m=0, key1stB_x_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[^Z][1a]__[M-_][df]__[M-^X][98]__[d][64]__[,][2c]__[&][26]_
Recovered Key(EP-1):_[x][78]__[U][55]__[s][73]__[B][42]__[3][33]__[G][47]_
Chain SP:_[x][78]__[U][55]__[s][73]__[B][42]__[3][33]__[G][47]_
Orig CT:_[s][73]__[M-^O][8f]__[T][54]__[A][41]__[^@][00]__[M-^Z][9a]_
Orig Key:_[x][78]__[U][55]__[s][73]__[B][42]__[3][33]__[G][47]_
CT 4 verify:_[s][73]__[M-^O][8f]__[T][54]__[A][41]__[^@][00]__[M-^Z][9a]_
We have Verification also.
Limits: nednodes=128:ppn=8,nodes=128:ppn=8,walltime=04:00:00
Resources:     cput=1211:52:25,mem=209366100kb,vmem=394953244kb,walltime
   =01:16:40

LowD4096t1024c1072693248m4hrs_bestCvrg0.o45941097
45941097
Lead at t=1 core=1, m=0, key1stB_m_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[M-^]][9d]__[L][4c]__[M-y][f9]__[^Y][19]__[M-^D][84]__[X][58]_
Recovered Key(EP-1):_[m][6d]__[W][57]__[j][6a]__[W][57]__[P][50]__[R][52]_
Chain SP:_[m][6d]__[W][57]__[j][6a]__[W][57]__[P][50]__[R][52]_
Orig CT:_[M-^]][9d]__[^L][0c]__[]][5d]__[M-^V][96]__[M-J][ca]__[K][4b]_
Orig Key:_[m][6d]__[W][57]__[j][6a]__[W][57]__[P][50]__[R][52]_
CT 4 verify:_[M-^]][9d]__[^L][0c]__[]][5d]__[M-^V][96]__[M-J][ca]__[K][4b]
   _
We have Verification also.
Limits: nednodes=128:ppn=8,nodes=128:ppn=8,walltime=04:00:00
Resources:     cput=1211:49:21,mem=209360508kb,vmem=394954576kb,walltime
   =01:16:37

LowD4096t1024c1072693248m4hrs_bestCvrg0.o45941098
45941098
Lead at t=1 core=1, m=0, key1stB_E_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
```

```
Found CT:_[0][30]__[M-‘][e0]__[%][25]__[M-L][cc]__[^D][04]__[j][6a]_
Recovered Key(EP-1):_[E][45]__[U][55]__[C][43]__[W][57]__[c][63]__[8][38]_
Chain SP:_[E][45]__[U][55]__[C][43]__[W][57]__[c][63]__[8][38]_
Orig CT:_[M-A][c1]__[M-w][f7]__[M-H][c8]__[y][79]__[M-^^][9e]__[M-*][aa]_
Orig Key:_[E][45]__[U][55]__[C][43]__[W][57]__[c][63]__[8][38]_
CT 4 verify:_[M-A][c1]__[M-w][f7]__[M-H][c8]__[y][79]__[M-^^][9e]__[M-*][
   aa]_
We have Verification also.
Limits: nednodes=128:ppn=8,nodes=128:ppn=8,walltime=04:00:00
Resources:     cput=1211:53:38,mem=209348504kb,vmem=394953776kb,walltime
   =01:16:39

LowD4096t1024c1072693248m4hrs_bestCvrg0.o45941099
45941099
Lead at t=1 core=1, m=0, key1stB_l_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[M-^N][8e]__[Z][5a]__[M-Y][d9]__[M-9][b9]__[M-z][fa]__[M-,][ac]_
Recovered Key(EP-1):_[l][6c]__[m][6d]__[b][62]__[y][79]__[J][4a]__[9][39]_
Chain SP:_[l][6c]__[m][6d]__[b][62]__[y][79]__[J][4a]__[9][39]_
Orig CT:_[M-%][a5]__[M-^W][97]__[,][2c]__[1][31]__[M-b][e2]__[^@][00]_
Orig Key:_[l][6c]__[m][6d]__[b][62]__[y][79]__[J][4a]__[9][39]_
CT 4 verify:_[M-%][a5]__[M-^W][97]__[,][2c]__[1][31]__[M-b][e2]__[^@][00]_
We have Verification also.
Limits: nednodes=128:ppn=8,nodes=128:ppn=8,walltime=04:00:00
Resources:     cput=1211:52:04,mem=209362100kb,vmem=394962712kb,walltime
   =01:16:39

LowD4096t1024c1072693248m4hrs_bestCvrg0.o45941100
45941100
Lead at t=1 core=1, m=0, key1stB_T_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[M-^G][87]__[M-F][c6]__[^P][10]__[M-^Z][9a]__[M-P][d0]__[M-$][a4
   ]_
Recovered Key(EP-1):_[T][54]__[9][39]__[7][37]__[X][58]__[h][68]__[A][41]_
Chain SP:_[T][54]__[9][39]__[7][37]__[X][58]__[h][68]__[A][41]_
Orig CT:_[M-+][ab]__[M-^U][95]__[M-^C][83]__[M-f][e6]__[M-*][aa]__[^_][1f]
   _
Orig Key:_[T][54]__[9][39]__[7][37]__[X][58]__[h][68]__[A][41]_
```

```
CT 4 verify:_[M-+][ab]__[M-^U][95]__[M-^C][83]__[M-f][e6]__[M-*][aa]__[^_
   ][1f]_
We have Verification also.
Limits: neednodes=128:ppn=8,nodes=128:ppn=8,walltime=04:00:00
Resources:     cput=1211:44:55,mem=209361688kb,vmem=394964420kb,walltime
   =01:16:36

LowD4096t1024c1072693248m4hrs_bestCvrg0.o45941101
45941101
Lead at t=1 core=1, m=0, key1stB_n_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[    ][09]__[M-w][f7]__[M-?][bf]__[M-n][ee]__[M-4][b4]__[M-A][c1]
   _
Recovered Key(EP-1):_[n][6e]__[Q][51]__[Q][51]__[5][35]__[v][76]__[d][64]_
Chain SP:_[n][6e]__[Q][51]__[Q][51]__[5][35]__[v][76]__[d][64]_
Orig CT:_[M-y][f9]__[b][62]__[j][6a]__[M-F][c6]__[M-^?][ff]__[M-^F][86]_
Orig Key:_[n][6e]__[Q][51]__[Q][51]__[5][35]__[v][76]__[d][64]_
CT 4 verify:_[M-y][f9]__[b][62]__[j][6a]__[M-F][c6]__[M-^?][ff]__[M-^F
   ][86]_
We have Verification also.
Limits: neednodes=128:ppn=8,nodes=128:ppn=8,walltime=04:00:00
Resources:     cput=1211:45:08,mem=209351744kb,vmem=394956380kb,walltime
   =01:16:34

LowD4096t1024c1072693248m4hrs_bestCvrg0.o45941102
45941102
Lead at t=1 core=1, m=0, key1stB_H_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[ ][20]__[M-^Q][91]__[M-^A][81]__[M-;][bb]__[M-‘][e0]__[M-x][f8]
   _
Recovered Key(EP-1):_[H][48]__[N][4e]__[J][4a]__[B][42]__[r][72]__[L][4c]_
Chain SP:_[H][48]__[N][4e]__[J][4a]__[B][42]__[r][72]__[L][4c]_
Orig CT:_[M-^W][97]__[M-s][f3]__[M-n][ee]__[p][70]__[M-~][fe]__[M-(][a8]_
Orig Key:_[H][48]__[N][4e]__[J][4a]__[B][42]__[r][72]__[L][4c]_
CT 4 verify:_[M-^W][97]__[M-s][f3]__[M-n][ee]__[p][70]__[M-~][fe]__[M-(][
   a8]_
We have Verification also.
Limits: neednodes=128:ppn=8,nodes=128:ppn=8,walltime=04:00:00
```

```
Resources:     cput=1211:55:33,mem=209347408kb,vmem=394948344kb,walltime
   =01:16:34

LowD4096t1024c1072693248m4hrs_bestCvrg0.o45941103
45941103
Lead at t=1 core=1, m=0, key1stB_k_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[M-.][ae]__[M-,][ac]__[M-]][dd]__()][29]__[M-p][f0]__[M-e][e5]_
Recovered Key(EP-1):_[k][6b]__[H][48]__[M][4d]__[W][57]__[F][46]__[M][4d]_
Chain SP:_[k][6b]__[H][48]__[M][4d]__[W][57]__[F][46]__[M][4d]_
Orig CT:_['][27]__[^K][0b]__[V][56]__[M-^M][8d]__[2][32]__[j][6a]_
Orig Key:_[k][6b]__[H][48]__[M][4d]__[W][57]__[F][46]__[M][4d]_
CT 4 verify:_['][27]__[^K][0b]__[V][56]__[M-^M][8d]__[2][32]__[j][6a]_
We have Verification also.
Limits: nednodes=128:ppn=8,nodes=128:ppn=8,walltime=04:00:00
Resources:     cput=1211:44:43,mem=209350384kb,vmem=394951400kb,walltime
   =01:16:44

LowD4096t1024c1072693248m4hrs_bestCvrg0.o45941104
45941104
Lead at t=1 core=1, m=0, key1stB_v_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[p][70]__[S][53]__[^_][1f]__[C][43]__[^B][02]__[1][31]_
Recovered Key(EP-1):_[v][76]__[3][33]__[K][4b]__[U][55]__[c][63]__[H][48]_
Chain SP:_[v][76]__[3][33]__[K][4b]__[U][55]__[c][63]__[H][48]_
Orig CT:_[^O][0f]__[M-Z][da]__[\][5c]__[$][24]__[M-^V][96]__[;][3b]_
Orig Key:_[v][76]__[3][33]__[K][4b]__[U][55]__[c][63]__[H][48]_
CT 4 verify:_[^O][0f]__[M-Z][da]__[\][5c]__[$][24]__[M-^V][96]__[;][3b]_
We have Verification also.
Limits: nednodes=128:ppn=8,nodes=128:ppn=8,walltime=04:00:00
Resources:     cput=1211:50:19,mem=209350572kb,vmem=394949672kb,walltime
   =01:16:41

LowD4096t1024c,1,072,693,248m 4hrs_bestCvrg0.o45941105
45941105
Lead at t=1 core=1, m=0, key1stB_N_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[^Z][1a]__[V][56]__[M-)][a9]__[m][6d]__[c][63]__[^P][10]_
```

```
Recovered Key(EP-1):_[N][4e]__[7][37]__[1][31]__[4][34]__[s][73]__[N][4e]_
Chain SP:_[N][4e]__[7][37]__[1][31]__[4][34]__[s][73]__[N][4e]_
Orig CT:_[7][37]__[M--][ad]__[D][44]__[M-@][c0]__[k][6b]__[d][64]_
Orig Key:_[N][4e]__[7][37]__[1][31]__[4][34]__[s][73]__[N][4e]_
CT 4 verify:_[7][37]__[M--][ad]__[D][44]__[M-@][c0]__[k][6b]__[d][64]_
We have Verification also.
Limits: nednodes=128:ppn=8,nodes=128:ppn=8,walltime=04:00:00
Resources:     cput=1211:52:59,mem=209361244kb,vmem=394961884kb,walltime
   =01:16:34

=====================================================================


LowD4096t1024c 2,145,386,496m 4hrs_bestCvrg0.o45941106
45941106
Limits: nednodes=128:ppn=8,nodes=128:ppn=8,walltime=04:00:00
Resources:     cput=2374:41:39,mem=238693212kb,vmem=424301964kb,walltime
   =02:30:09

LowD4096t 1024c 2145386496m4hrs_bestCvrg0.o45941107
45941107
Limits: nednodes=128:ppn=8,nodes=128:ppn=8,walltime=04:00:00
Resources: cput=2374:42:05,mem=238692568kb,vmem=424302016kb,walltime
   =02:30:09

LowD4096t1024c 2145386496m4hrs_bestCvrg0.o45941109
45941109
Limits: nednodes=128:ppn=8,nodes=128:ppn=8,walltime=04:00:00
Resources: cput=2375:02:19,mem=238682140kb,vmem=424295860kb,walltime
   =02:30:10

LowD4096t1024c 2145386496m4hrs_bestCvrg0.o45941110
45941110
Limits: nednodes=128:ppn=8,nodes=128:ppn=8,walltime=04:00:00
Resources: cput=2374:54:07,mem=238687068kb,vmem=424296816kb,walltime
   =02:30:19

LowD4096t1024c 2145386496m4hrs_bestCvrg0.o45941111
```

```
45941111
Limits: nednodes=128:ppn=8,nodes=128:ppn=8,walltime=04:00:00
Resources: cput=2374:57:30,mem=238696304kb,vmem=424303164kb,walltime
   =02:30:47

LowD4096t1024c  2145386496m4hrs_bestCvrg0.o45941113
45941113
Limits: nednodes=128:ppn=8,nodes=128:ppn=8,walltime=04:00:00
Resources: cput=2374:37:16,mem=238698680kb,vmem=424295704kb,walltime
   =02:30:12

LowD4096t1024c 2145386496m4hrs_bestCvrg0.o45941115
45941115
Limits: nednodes=128:ppn=8,nodes=128:ppn=8,walltime=04:00:00
Resources: cput=2374:55:03,mem=238698196kb,vmem=424295848kb,walltime
   =02:30:10

LowD4096t1024c2145386496m4hrs_bestCvrg0.o45941116
45941116
Limits: nednodes=128:ppn=8,nodes=128:ppn=8,walltime=04:00:00
Resources: cput=2374:47:03,mem=238699648kb,vmem=424296820kb,walltime
   =02:30:06

============================================================

LowD4096t1024c2145386496m4hrs_bestCvrg0.o45941114
45941114
=>> PBS: job killed: walltime 14406 exceeded limit 14400
Limits: nednodes=128:ppn=8,nodes=128:ppn=8,walltime=04:00:00
Resources:     cput=2364:49:38,mem=238634156kb,vmem=424294624kb,walltime
   =04:00:12

LowD4096t1024c2145386496m4hrs_bestCvrg0.o45941112
45941112
=>> PBS: job killed: walltime 14412 exceeded limit 14400
Limits: nednodes=128:ppn=8,nodes=128:ppn=8,walltime=04:00:00
Resources:     cput=2358:16:31,mem=238583224kb,vmem=424294648kb,walltime
   =04:00:13
```

The whole batch where m= 2,145,386,496 records and W = 4:00:00 hours ran out of time.

```
for j in 45941137 45941136 45941135 45941134 45941133 45941132 45941131
   45941130 45941129 45941128 45941127 45941125 45941124 45941123 45941122
    45941121 45941120 45941119 45941118 45941117 ; do ls *.o${j} ; echo $j
   ; cat -v *.[oe]$j* | egrep "Verification|walltime|ERR" ; echo "" ; done
    | sed -e "s/\t\t/\t/" | sed -e "s/Orig/\nOrig/g" | sed -e "s/Found/\
   nFound/" | sed -e "s/Recovered/\nRecovered/" | sed -e "s/Yaaaay l/L/" |
    sed -e "s/Chain/\nChain/" | sed -e "s/CT 4 verify/\nCT 4 verify/" |
   sed -e "s/Yaaaay: /\n/"

LowD1048576t1024c2145386496m4hrs_bestCvrg7.o45941137
45941137
=>> PBS: job killed: walltime 14432 exceeded limit 14400
Limits: neednodes=128:ppn=8,nodes=128:ppn=8,walltime=04:00:00
Resources:     cput=4098:09:55,mem=223845120kb,vmem=423044124kb,walltime
   =04:00:32

LowD1048576t1024c2145386496m4hrs_bestCvrg7.o45941136
45941136
=>> PBS: job killed: walltime 14437 exceeded limit 14400
Limits: neednodes=128:ppn=8,nodes=128:ppn=8,walltime=04:00:00
Resources:     cput=4101:15:06,mem=223834152kb,vmem=423044124kb,walltime
   =04:00:38

LowD1048576t1024c2145386496m4hrs_bestCvrg7.o45941135
45941135
=>> PBS: job killed: walltime 14404 exceeded limit 14400
Limits: neednodes=128:ppn=8,nodes=128:ppn=8,walltime=04:00:00
Resources:     cput=4090:57:43,mem=223850092kb,vmem=423044040kb,walltime
   =04:00:04

LowD1048576t1024c2145386496m4hrs_bestCvrg7.o45941134
45941134
=>> PBS: job killed: walltime 14435 exceeded limit 14400
```

```
Limits: neednodes=128:ppn=8,nodes=128:ppn=8,walltime=04:00:00
Resources:     cput=4099:24:41,mem=223846932kb,vmem=423044996kb,walltime
   =04:00:35

LowD1048576t1024c2145386496m4hrs_bestCvrg7.o45941133
45941133
=>> PBS: job killed: walltime 14408 exceeded limit 14400
Limits: neednodes=128:ppn=8,nodes=128:ppn=8,walltime=04:00:00
Resources:     cput=4069:26:39,mem=223844880kb,vmem=423044008kb,walltime
   =04:00:09

LowD1048576t1024c2145386496m4hrs_bestCvrg7.o45941132
45941132
=>> PBS: job killed: walltime 14427 exceeded limit 14400
Limits: neednodes=128:ppn=8,nodes=128:ppn=8,walltime=04:00:00
Resources:     cput=4095:44:38,mem=223850592kb,vmem=423044140kb,walltime
   =04:00:27

LowD1048576t1024c2145386496m4hrs_bestCvrg7.o45941131
45941131
=>> PBS: job killed: walltime 14401 exceeded limit 14400
Limits: neednodes=128:ppn=8,nodes=128:ppn=8,walltime=04:00:00
Resources:     cput=4090:20:54,mem=223846276kb,vmem=423044920kb,walltime
   =04:00:02

LowD1048576t1024c2145386496m4hrs_bestCvrg7.o45941130
45941130
=>> PBS: job killed: walltime 14431 exceeded limit 14400
Limits: neednodes=128:ppn=8,nodes=128:ppn=8,walltime=04:00:00
Resources:     cput=4098:22:40,mem=223845612kb,vmem=423043980kb,walltime
   =04:00:31

LowD1048576t1024c2145386496m4hrs_bestCvrg7.o45941129
45941129
=>> PBS: job killed: walltime 14444 exceeded limit 14400
Limits: neednodes=128:ppn=8,nodes=128:ppn=8,walltime=04:00:00
Resources:     cput=4102:51:33,mem=223846120kb,vmem=423043904kb,walltime
   =04:00:44
```

```
LowD1048576t1024c2145386496m4hrs_bestCvrg7.o45941128
45941128
=>> PBS: job killed: walltime 14412 exceeded limit 14400
Limits: neednodes=128:ppn=8,nodes=128:ppn=8,walltime=04:00:00
Resources:     cput=4092:21:32,mem=223846264kb,vmem=423044116kb,walltime
   =04:00:12
```

And so did the batch were m=1,072,693,248 records and W=4:00:00 hours.

```
LowD1048576t1024c1072693248m4hrs_bestCvrg3.o45941127
45941127
=>> PBS: job killed: walltime 14427 exceeded limit 14400
Limits: neednodes=128:ppn=8,nodes=128:ppn=8,walltime=04:00:00
Resources:     cput=4097:03:28,mem=200784084kb,vmem=393683992kb,walltime
   =04:00:27

LowD1048576t1024c1072693248m4hrs_bestCvrg3.o45941125
45941125
=>> PBS: job killed: walltime 14438 exceeded limit 14400
Limits: neednodes=128:ppn=8,nodes=128:ppn=8,walltime=04:00:00
Resources:     cput=4101:03:10,mem=200783436kb,vmem=393684236kb,walltime
   =04:00:38

LowD1048576t1024c1072693248m4hrs_bestCvrg3.o45941124
45941124
=>> PBS: job killed: walltime 14427 exceeded limit 14400
Limits: neednodes=128:ppn=8,nodes=128:ppn=8,walltime=04:00:00
Resources:     cput=4097:02:09,mem=200785656kb,vmem=393683848kb,walltime
   =04:00:27

LowD1048576t1024c1072693248m4hrs_bestCvrg3.o45941123
45941123
=>> PBS: job killed: walltime 14439 exceeded limit 14400
Limits: neednodes=128:ppn=8,nodes=128:ppn=8,walltime=04:00:00
Resources:     cput=4100:55:17,mem=200785360kb,vmem=393683988kb,walltime
   =04:00:39
```

```
LowD1048576t1024c1072693248m4hrs_bestCvrg3.o45941122
45941122
=>> PBS: job killed: walltime 14413 exceeded limit 14400
Limits: neednodes=128:ppn=8,nodes=128:ppn=8,walltime=04:00:00
Resources:     cput=4093:57:13,mem=200783872kb,vmem=393683952kb,walltime
   =04:00:13

LowD1048576t1024c1072693248m4hrs_bestCvrg3.o45941121
45941121
=>> PBS: job killed: walltime 14433 exceeded limit 14400
Limits: neednodes=128:ppn=8,nodes=128:ppn=8,walltime=04:00:00
Resources:     cput=4099:42:51,mem=200784836kb,vmem=393684052kb,walltime
   =04:00:33

LowD1048576t1024c1072693248m4hrs_bestCvrg3.o45941120
45941120
=>> PBS: job killed: walltime 14434 exceeded limit 14400
Limits: neednodes=128:ppn=8,nodes=128:ppn=8,walltime=04:00:00
Resources:     cput=4099:33:37,mem=200785424kb,vmem=393683808kb,walltime
   =04:00:34

LowD1048576t1024c1072693248m4hrs_bestCvrg3.o45941119
45941119
=>> PBS: job killed: walltime 14426 exceeded limit 14400
Limits: neednodes=128:ppn=8,nodes=128:ppn=8,walltime=04:00:00
Resources:     cput=4098:01:41,mem=200784492kb,vmem=393683988kb,walltime
   =04:00:26

LowD1048576t1024c1072693248m4hrs_bestCvrg3.o45941118
45941118
=>> PBS: job killed: walltime 14443 exceeded limit 14400
Limits: neednodes=128:ppn=8,nodes=128:ppn=8,walltime=04:00:00
Resources:     cput=4100:38:40,mem=200785156kb,vmem=393683772kb,walltime
   =04:00:43

LowD1048576t1024c1072693248m4hrs_bestCvrg3.o45941117
45941117
=>> PBS: job killed: walltime 14421 exceeded limit 14400
```

```
Limits: neednodes=128:ppn=8,nodes=128:ppn=8,walltime=04:00:00
Resources:     cput=4097:28:41,mem=200783480kb,vmem=393683896kb,walltime
   =04:00:21
```

A.2.3 Decreasing output density further

We attempted to radically decrease the debug information especially for the Child processes. Job ID 46040582 46040583 46040584 46040585

```
The number of booked cores is 1024 (1 Parent and 1023 children)
plaintext is: PTPTPT
Random key: (multiple) and it is used as the start point of the first
   chain.
After encrypting the plaintext T times (1048576 = 2^20, ) the chain end
   point becomes : (multiple)
Ciphertext Challenge: (multiple)
CT decrypts back to: (multiple)
output log: (multiple)
Output size (printf overhead): about 3170 lines, 323422 characters.
error log: (multiple)
Job log: (multiple)
source code file:(multiple)
job submission script:(multiple)
object file:(multiple)
Cores: 1024
usableCores: 1023
PID: (multiple)
Date: (multiple)
M= 1,072,693,248 = 2^30
C= 1024
W= 4 hours
T= 1048576 =2^20
B= 48
E= 2
H= 0
L= 0
```

```
S= 1
R= GPC
notes=
```

Observation: Runs that had t= 1048576 = 2^20 iterations, and M= 1,072,693,248 = 2^30 records ran out of time (4 hours walltime).

```
VLowD 1048576t 1024c 1072693248m 4hrs_bestCvrg3.o46040582
46040582
=>> PBS: job killed: walltime 14408 exceeded limit 14400
Limits: neednodes=128:ppn=8,nodes=128:ppn=8,walltime=04:00:00
Resources:     cput=4092:27:06,mem=200778512kb,vmem=393683536kb,walltime
   =04:00:08

VLowD 1048576t 1024c 1072693248m 4hrs_bestCvrg3.o46040583
46040583
=>> PBS: job killed: walltime 14443 exceeded limit 14400
Limits: neednodes=128:ppn=8,nodes=128:ppn=8,walltime=04:00:00
Resources:     cput=4101:31:34,mem=200777256kb,vmem=393683928kb,walltime
   =04:00:43

VLowD 1048576t 1024c 1072693248m 4hrs_bestCvrg3.o46040584
46040584
=>> PBS: job killed: walltime 14432 exceeded limit 14400
Limits: neednodes=128:ppn=8,nodes=128:ppn=8,walltime=04:00:00
Resources:     cput=4099:17:11,mem=200775864kb,vmem=393683684kb,walltime
   =04:00:33

VLowD 1048576t 1024c 1072693248m 4hrs_bestCvrg3.o46040585
46040585
=>> PBS: job killed: walltime 14404 exceeded limit 14400
Limits: neednodes=128:ppn=8,nodes=128:ppn=8,walltime=04:00:00
Resources:     cput=4090:21:49,mem=200774908kb,vmem=393683820kb,walltime
   =04:00:05
```

Job ID 46040586 46040587 46040588 46040589

```
The number of booked cores is 1024 (1 Parent and 1023 children)
```

```
plaintext is: PTPTPT
Random key: (multiple) and it is used as the start point of the first
   chain.
After encrypting the plaintext T times (1048576 = 2^20, ) the chain end
   point becomes : (multiple)
Ciphertext Challenge: (multiple)
CT decrypts back to: (multiple)
output log: (multiple)
Output size (printf overhead): about 3170 lines, 323422 characters.
error log: (multiple)
Job log: (multiple)
source code file:(multiple)
job submission script:(multiple)
object file:(multiple)
Cores: 1024
usableCores: 1023
PID: (multiple)
Date: (multiple)
M= M= 2,145,386,496 = 2^31 records
C= 1024
W= 4 hours
T= 1048576 = 2^20 iterations
B= 48
E= 2
H= 0
L= 0
S= 1
R= GPC
notes=
```

Observation: So did fail the runs that had t= 1048576 = 2^20 iterations, and M= 2,145,386,496 = 2^31 records.

```
VLowD 1048576t 1024c 2145386496m 4hrs_bestCvrg7.o46040586
46040586
=>> PBS: job killed: walltime 14422 exceeded limit 14400
Limits: nednodes=128:ppn=8,nodes=128:ppn=8,walltime=04:00:00
Resources:    cput=4094:39:31,mem=223837292kb,vmem=423044224kb,walltime
```

```
   =04:00:23

VLowD 1048576t 1024c 2145386496m 4hrs_bestCvrg7.o46040587
46040587
=>> PBS: job killed: walltime 14418 exceeded limit 14400
Limits: nednodes=128:ppn=8,nodes=128:ppn=8,walltime=04:00:00
Resources:     cput=4094:26:06,mem=223839788kb,vmem=423043692kb,walltime
   =04:00:18

VLowD 1048576t 1024c 2145386496m 4hrs_bestCvrg7.o46040588
46040588
=>> PBS: job killed: walltime 14409 exceeded limit 14400
Limits: nednodes=128:ppn=8,nodes=128:ppn=8,walltime=04:00:00
Resources:     cput=4091:07:03,mem=223837044kb,vmem=423043964kb,walltime
   =04:00:09

VLowD 1048576t 1024c 2145386496m 4hrs_bestCvrg7.o46040589
46040589
=>> PBS: job killed: walltime 14419 exceeded limit 14400
Limits: nednodes=128:ppn=8,nodes=128:ppn=8,walltime=04:00:00
Resources:     cput=4095:23:30,mem=223823884kb,vmem=423043764kb,walltime
   =04:00:19
```

Job ID 46040598 46040599

```
The number of booked cores is 1024 (1 Parent and 1023 children)
plaintext is: PTPTPT
Random key: (multiple) and it is used as the start point of the first
   chain.
After encrypting the plaintext T times (1048576 = 2^20, ) the chain end
   point becomes : (multiple)
Ciphertext Challenge: (multiple)
CT decrypts back to: (multiple)
output log: (multiple)
Output size (printf overhead): about 3170 lines, 323422 characters.
error log: (multiple)
Job log: (multiple)
```

```
source code file:(multiple)
job submission script:(multiple)
object file:(multiple)
Cores: 1024
usableCores: 1023
PID: (multiple)
Date: (multiple)
M= M= 1,072,693,248 = 2^30 records
C= 1024
W= 4 hours
T= 1048576 = 2^20 iterations
B= 48
E= 2
H= 0
L= 0
S= 1
R= GPC
notes=
```

Observation: Some 8 hours run also ran out of time even with a smaller m of 1,072,693,248 records.

```
VLowD 1048576t 1024c 1072693248m 8hrs_bestCvrg3.o46040598
46040598
=>> PBS: job killed: walltime 28825 exceeded limit 28800
Limits: nednodes=128:ppn=8,nodes=128:ppn=8,walltime=08:00:00
Resources:     cput=8191:44:42,mem=200861044kb,vmem=393683760kb,walltime
   =08:00:25

VLowD 1048576t 1024c 1072693248m 8hrs_bestCvrg3.o46040599
46040599
=>> PBS: job killed: walltime 28840 exceeded limit 28800
Limits: nednodes=128:ppn=8,nodes=128:ppn=8,walltime=08:00:00
Resources:     cput=8196:00:24,mem=200859644kb,vmem=393684116kb,walltime
   =08:00:40
```

Job ID 46040572

```
The number of booked cores is 1024 (1 Parent and 1023 children)
plaintext is: PTPTPT
Random key: (multiple) and it is used as the start point of the first
   chain.
After encrypting the plaintext T times (4096 = 2^12, ) the chain end point
    becomes : (multiple)
Ciphertext Challenge: (multiple)
CT decrypts back to: (multiple)
output log: (multiple)
Output size (printf overhead): about 3170 lines, 323422 characters.
error log: (multiple)
Job log: (multiple)
source code file:(multiple)
job submission script:(multiple)
object file:(multiple)
Cores: 1024
usableCores: 1023
PID: (multiple)
Date: (multiple)
M= 1,072,693,248 = 2^30 records
C= 1024
W= 4 hours
T= 4096 = 2^12 iterations
B= 48
E= 2
H= 0
L= 0
S= 1
R= GPC
notes=
```

Observation: With 4096 T, 4 hours run were mostly successful. Job 46040572 finished in 1 hour 11 minutes, found the Key but also found 5 false positive leads. Two of the keys were 0x6cc94a0fb8fe, Two were 0x3173e31408b3, One was 0xcb59825b9174. Those false positives were in 5 different chains and 2 pairs are of the same values. This shows that there is collision between chains.

```
VLowD 4096t 1024c 1072693248m 4hrs_bestCvrg0.o46040572
```

```
============================================================
46040572
Lead at t=2503 core=400, m=1217394, key1stB_l_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[M-E][c5]__[M- ][a0]__[M-O][cf]__[e][65]__[M-s][f3]__[$][24]_
Recovered Key(EP-1):_[l][6c]__[M-I][c9]__[J][4a]__[^O][0f]__[M-8][b8]__[M
   -~][fe]_
Chain SP:_[M-o][ef]__[^G][07]__[M-^P][90]__[M-B][c2]__[O][4f]__[M-^G][87]_
Orig CT:_[=][3d]__[M-^I][89]__[i][69]__[M-^^][9e]__[M-3][b3]__[M-b][e2]_
Orig Key:_[H][48]__[O][4f]__[E][45]__[P][50]__[S][53]__[q][71]_
CT 4 verify:_[ ][20]__[M-d][e4]__[P][50]__[M-^^][9e]__[M-Z][da]__[D][44]_

Lead at t=2246 core=853, m=1633008, key1stB_l_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[M-=][bd]__[A][41]__[M-~][fe]__[^L][0c]__[^M][0d]__[M-=][bd]_
Recovered Key(EP-1):_[l][6c]__[M-I][c9]__[J][4a]__[^O][0f]__[M-8][b8]__[M
   -~][fe]_
Chain SP:_[M-k][eb]__[D][44]__[M-6][b6]__[`][60]__[M-e][e5]__[J][4a]_
Orig CT:_[=][3d]__[M-^I][89]__[i][69]__[M-^^][9e]__[M-3][b3]__[M-b][e2]_
Orig Key:_[H][48]__[O][4f]__[E][45]__[P][50]__[S][53]__[q][71]_
CT 4 verify:_[ ][20]__[M-d][e4]__[P][50]__[M-^^][9e]__[M-Z][da]__[D][44]_

Lead at t=1199 core=1010, m=1650234, key1stB_1_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[M-z][fa]__[o][6f]__[8][38]__[M-5][b5]__[*][2a]__[M-_][df]_
Recovered Key(EP-1):_[1][31]__[s][73]__[M-c][e3]__[^T][14]__[^H][08]__[M
   -3][b3]_
Chain SP:_[^_][1f]__[M-~][fe]__[^S][13]__[5][35]__[t][74]__[M-^N][8e]_
Orig CT:_[=][3d]__[M-^I][89]__[i][69]__[M-^^][9e]__[M-3][b3]__[M-b][e2]_
Orig Key:_[H][48]__[O][4f]__[E][45]__[P][50]__[S][53]__[q][71]_
CT 4 verify:_[J][4a]__[M-^I][89]__[Q][51]__[!][21]__[M-^I][89]__[<][3c]_

Lead at t=1049 core=1007, m=1578552, key1stB_1_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[J][4a]__[M-!][a1]__[M-@][c0]__[M-`][e0]__[z][7a]__[M-d][e4]_
Recovered Key(EP-1):_[1][31]__[s][73]__[M-c][e3]__[^T][14]__[^H][08]__[M
   -3][b3]_
Chain SP:_[E][45]__[M-~][fe]__[M-@][c0]__[-][2d]__[M-*][aa]__[M-<][bc]_
```

```
Orig CT:_[=][3d]__[M-^I][89]__[i][69]__[M-^^][9e]__[M-3][b3]__[M-b][e2]_
Orig Key:_[H][48]__[O][4f]__[E][45]__[P][50]__[S][53]__[q][71]_
CT 4 verify:_[J][4a]__[M-^I][89]__[Q][51]__[!][21]__[M-^I][89]__[<][3c]_

Lead at t=432 core=141, m=487104, key1stB_M-K_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[Z][5a]__[b][62]__[^C][03]__[M-m][ed]__[M- ][a0]__[M-x][f8]_
Recovered Key(EP-1):_[M-K][cb]__[Y][59]__[M-^B][82]__[[][5b]__[M-^Q][91]__
   [t][74]_
Chain SP:_[t][74]__[*][2a]__[M-g][e7]__[M-;][bb]__[^F][06]__[U][55]_
Orig CT:_[=][3d]__[M-^I][89]__[i][69]__[M-^^][9e]__[M-3][b3]__[M-b][e2]_
Orig Key:_[H][48]__[O][4f]__[E][45]__[P][50]__[S][53]__[q][71]_
CT 4 verify:_[k][6b]__[^][5e]__[M-Q][d1]__[M-B][c2]__[J][4a]__[2][32]_

Lead at t=1 core=1, m=0, key1stB_H_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[M-_][df]__[M-h][e8]__[T][54]__[+][2b]__[M-T][d4]__[j][6a]_
Recovered Key(EP-1):_[H][48]__[O][4f]__[E][45]__[P][50]__[S][53]__[q][71]_
Chain SP:_[H][48]__[O][4f]__[E][45]__[P][50]__[S][53]__[q][71]_
Orig CT:_[=][3d]__[M-^I][89]__[i][69]__[M-^^][9e]__[M-3][b3]__[M-b][e2]_
Orig Key:_[H][48]__[O][4f]__[E][45]__[P][50]__[S][53]__[q][71]_
CT 4 verify:_[=][3d]__[M-^I][89]__[i][69]__[M-^^][9e]__[M-3][b3]__[M-b][e2
   ]_
We have Verification also.

Limits: nednodes=128:ppn=8,nodes=128:ppn=8,walltime=04:00:00
Resources:     cput=1211:07:03,mem=209196776kb,vmem=394828052kb,walltime
   =01:11:28
```

Job ID 46040573

```
The number of booked cores is 1024 (1 Parent and 1023 children)
plaintext is: PTPTPT
Random key: (multiple) and it is used as the start point of the first
   chain.
After encrypting the plaintext T times (1048576 = 2^20, ) the chain end
   point becomes : (multiple)
```

```
Ciphertext Challenge: (multiple)
CT decrypts back to: (multiple)
output log: (multiple)
Output size (printf overhead): about 3170 lines, 323422 characters.
error log: (multiple)
Job log: (multiple)
source code file:(multiple)
job submission script:(multiple)
object file:(multiple)
Cores: 1024
usableCores: 1023
PID: (multiple)
Date: (multiple)
M= 1,072,693,248 = 2^30 records
C= 1024
W= 4 hours
T= 4096 = 2^12 iterations
B= 48
E= 2
H= 0
L= 0
S= 1
R= GPC
notes=VLowD4096t1024c1072693248m4hrs_bestCvrg0
```

Observation: With the same 4096 T, 4 hours, job 46040573 also finished in about 1 hour 11 minutes, found the Key, and also found 9 false positive leads. Six of the false positives were all 0xcc192590b40f which means a high level of collision. There is currently no way to tell exactly what is the percentage of the collision because we do not record it. We only record those that flagged as potential lead and by definition they are a small percentage. So we can derive that the overall collision is equal to or larger than the lead collision. The three other false alarms are unique.

```
VLowD 4096t 1024c 1072693248m 4hrs_bestCvrg0.o46040573
46040573
Lead at t=3264 core=379, m=1247274, key1stB_M-L_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[M-,][ac]__[^A][01]__[^Z][1a]__[y][79]__[L][4c]__[M-:][ba]_
```

```
Recovered Key(EP-1):_[M-L][cc]__[^Y][19]__[%][25]__[M-^P][90]__[M-4][b4]__
    [^O][0f]_
Chain SP:_[9][39]__[t][74]__[(][28]__[M-^D][84]__[M-1][b1]__[o][6f]_
Orig CT:_[M-‘][e0]__[M-2][b2]__[M-o][ef]__[M-"][a2]__[M-}][fd]__[^@][00]_
Orig Key:_[b][62]__[u][75]__[C][43]__[z][7a]__[S][53]__[l][6c]_
CT 4 verify:_[M-}][fd]__[M-8][b8]__[^@][00]__[^][5e]__[V][56]__[M-,][ac]_

Lead at t=2924 core=739, m=1362978, key1stB_M-L_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[~][7e]__[M-T][d4]__[ ][09]__[V][56]__[9][39]__[M-L][cc]_
Recovered Key(EP-1):_[M-L][cc]__[^Y][19]__[%][25]__[M-^P][90]__[M-4][b4]__
    [^O][0f]_
Chain SP:_[M-^W][97]__[M-^]][9d]__[M-^Y][99]__[M-{][fb]__[M-F][c6]__[^?][7
    f]_
Orig CT:_[M-‘][e0]__[M-2][b2]__[M-o][ef]__[M-"][a2]__[M-}][fd]__[^@][00]_
Orig Key:_[b][62]__[u][75]__[C][43]__[z][7a]__[S][53]__[l][6c]_
CT 4 verify:_[M-}][fd]__[M-8][b8]__[^@][00]__[^][5e]__[V][56]__[M-,][ac]_

Lead at t=2199 core=29, m=983826, key1stB_M-L_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[b][62]__[8][38]__[M-#][a3]__[M-^G][87]__[.][2e]__[M-\][dc]_
Recovered Key(EP-1):_[M-L][cc]__[^Y][19]__[%][25]__[M-^P][90]__[M-4][b4]__
    [^O][0f]_
Chain SP:_[^R][12]__[M-w][f7]__[q][71]__[M-^]][9d]__[B][42]__[y][79]_
Orig CT:_[M-‘][e0]__[M-2][b2]__[M-o][ef]__[M-"][a2]__[M-}][fd]__[^@][00]_
Orig Key:_[b][62]__[u][75]__[C][43]__[z][7a]__[S][53]__[l][6c]_
CT 4 verify:_[M-}][fd]__[M-8][b8]__[^@][00]__[^][5e]__[V][56]__[M-,][ac]_

Lead at t=2198 core=28, m=1278660, key1stB_M-L_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[r][72]__[]][5d]__[M-\][dc]__[M-^I][89]__[M-}][fd]__[M-[][db]_
Recovered Key(EP-1):_[M-L][cc]__[^Y][19]__[%][25]__[M-^P][90]__[M-4][b4]__
    [^O][0f]_
Chain SP:_[M-e][e5]__[ ][20]__[1][31]__[{][7b]__[M-B][c2]__[4][34]_
Orig CT:_[M-‘][e0]__[M-2][b2]__[M-o][ef]__[M-"][a2]__[M-}][fd]__[^@][00]_
Orig Key:_[b][62]__[u][75]__[C][43]__[z][7a]__[S][53]__[l][6c]_
CT 4 verify:_[M-}][fd]__[M-8][b8]__[^@][00]__[^][5e]__[V][56]__[M-,][ac]_
```

```
Lead at t=1470 core=769, m=32166, key1stB_M-L_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[=][3d]__[M-f][e6]__[^W][17]__[M-^[][9b]__[M-]][dd]__[M-4][b4]_
Recovered Key(EP-1):_[M-L][cc]__[^Y][19]__[%][25]__[M-^P][90]__[M-4][b4]__
   [^O][0f]_
Chain SP:_[M-f][e6]__[M-z][fa]__[M-W][d7]__[^L][0c]__[‘][60]__[(][28]_
Orig CT:_[M-‘][e0]__[M-2][b2]__[M-o][ef]__[M-"][a2]__[M-}][fd]__[^@][00]_
Orig Key:_[b][62]__[u][75]__[C][43]__[z][7a]__[S][53]__[l][6c]_
CT 4 verify:_[M-}][fd]__[M-8][b8]__[^@][00]__[^][5e]__[V][56]__[M-,][ac]_

Lead at t=959 core=818, m=1085934, key1stB_M-L_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[^K][0b]__[G][47]__[M-^O][8f]__[M-X][d8]__[M-W][d7]__[M-o][ef]_
Recovered Key(EP-1):_[M-L][cc]__[^Y][19]__[%][25]__[M-^P][90]__[M-4][b4]__
   [^O][0f]_
Chain SP:_[^H][08]__[M-~][fe]__[M-S][d3]__[x][78]__[>][3e]__[‘][60]_
Orig CT:_[M-‘][e0]__[M-2][b2]__[M-o][ef]__[M-"][a2]__[M-}][fd]__[^@][00]_
Orig Key:_[b][62]__[u][75]__[C][43]__[z][7a]__[S][53]__[l][6c]_
CT 4 verify:_[M-}][fd]__[M-8][b8]__[^@][00]__[^][5e]__[V][56]__[M-,][ac]_

Lead at t=226 core=201, m=801672, key1stB_A_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[=][3d]__[^F][06]__[M-"][a2]__[m][6d]__[^W][17]__[M-j][ea]_
Recovered Key(EP-1):_[A][41]__[M-^V][96]__[^V][16]__[^W][17]__[M-w][f7]__
   [^N][0e]_
Chain SP:_[n][6e]__[M-d][e4]__[M-K][cb]__[Z][5a]__[@][40]__[M-^S][93]_
Orig CT:_[M-‘][e0]__[M-2][b2]__[M-o][ef]__[M-"][a2]__[M-}][fd]__[^@][00]_
Orig Key:_[b][62]__[u][75]__[C][43]__[z][7a]__[S][53]__[l][6c]_
CT 4 verify:_()][29]__[M-C][c3]__[M-^O][8f]__[V][56]__[M-@][c0]__[M-{][fb]
   _

Lead at t=155 core=1015, m=764166, key1stB_F_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[M][4d]__[c][63]__[}][7d]__[M-’][a7]__[M-^S][93]__[K][4b]_
Recovered Key(EP-1):_[F][46]__[M-1][b1]__[v][76]__[M-y][f9]__[M-_][df]__[M
   -^?][ff]_
Chain SP:_[R][52]__[8][38]__[M-w][f7]__[M-^V][96]__[k][6b]__[M-;][bb]_
Orig CT:_[M-‘][e0]__[M-2][b2]__[M-o][ef]__[M-"][a2]__[M-}][fd]__[^@][00]_
```

```
Orig Key:_[b][62]__[u][75]__[C][43]__[z][7a]__[S][53]__[l][6c]_
CT 4 verify:_[^_][1f]__[M-l][ec]__[]][5d]__[M-:][ba]__[y][79]__[M-o][ef]_

Lead at t=1 core=1, m=0, key1stB_b_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[M-o][ef]__[M-m][ed]__[g][67]__[M-G][c7]__[A][41]__[^K][0b]_
Recovered Key(EP-1):_[b][62]__[u][75]__[C][43]__[z][7a]__[S][53]__[l][6c]_
Chain SP:_[b][62]__[u][75]__[C][43]__[z][7a]__[S][53]__[l][6c]_
Orig CT:_[M-‘][e0]__[M-2][b2]__[M-o][ef]__[M-"][a2]__[M-}][fd]__[^@][00]_
Orig Key:_[b][62]__[u][75]__[C][43]__[z][7a]__[S][53]__[l][6c]_
CT 4 verify:_[M-‘][e0]__[M-2][b2]__[M-o][ef]__[M-"][a2]__[M-}][fd]__[^@
   ][00]_
We have Verification also.

Lead at t=846 core=980, m=2047308, key1stB_M- _
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[M-;][bb]__[M-^]][9d]__[M-L][cc]__[i][69]__[^_][1f]__[7][37]_
Recovered Key(EP-1):_[M- ][a0]__[V][56]__[V][56]__[N][4e]__[M-^M][8d]__[^_
   ][1f]_
Chain SP:_[M-O][cf]__[W][57]__[M-D][c4]__[u][75]__[M-c][e3]__[M-^Z][9a]_
Orig CT:_[M-‘][e0]__[M-2][b2]__[M-o][ef]__[M-"][a2]__[M-}][fd]__[^@][00]_
Orig Key:_[b][62]__[u][75]__[C][43]__[z][7a]__[S][53]__[l][6c]_
CT 4 verify:_[M-c][e3]__[M-c][e3]__[8][38]__[:][3a]__[C][43]__[M-W][d7]_

Limits: nednodes=128:ppn=8,nodes=128:ppn=8,walltime=04:00:00
Resources:     cput=1211:10:09,mem=209203424kb,vmem=394832964kb,walltime
   =01:11:29
```

Job ID 46040574

```
The number of booked cores is 1024 (1 Parent and 1023 children)
plaintext is: PTPTPT
Random key: (multiple) and it is used as the start point of the first
   chain.
After encrypting the plaintext T times (1048576 = 2^20, ) the chain end
   point becomes : (multiple)
Ciphertext Challenge: (multiple)
```

```
CT decrypts back to: (multiple)
output log: (multiple)
Output size (printf overhead): about 3170 lines, 323422 characters.
error log: (multiple)
Job log: (multiple)
source code file:(multiple)
job submission script:(multiple)
object file:(multiple)
Cores: 1024
usableCores: 1023
PID: (multiple)
Date: (multiple)
M= 1,072,693,248 = 2^30 records
C= 1024
W= 4 hours
T= 4096 = 2^12 iterations
B= 48
E= 2
H= 0
L= 0
S= 1
R= GPC
notes=VLowD4096t1024c1072693248m4hrs_bestCvrg0
```

Observation:Job 46040574 was successful without false positive leads.

```
VLowD 4096t 1024c 1072693248m 4hrs_bestCvrg0.o46040574
46040574
Lead at t=1 core=1, m=0, key1stB_s_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[^V][16]__[M-^M][8d]__[M-!][a1]__[M-W][d7]__[M-P][d0]__[M-^][de]
   _
Recovered Key(EP-1):_[s][73]__[B][42]__[b][62]__[q][71]__[a][61]__[L][4c]_
Chain SP:_[s][73]__[B][42]__[b][62]__[q][71]__[a][61]__[L][4c]_
Orig CT:_[M-h][e8]__[M-}][fd]__[M-d][e4]__[M-^F][86]__[m][6d]__[M-G][c7]_
Orig Key:_[s][73]__[B][42]__[b][62]__[q][71]__[a][61]__[L][4c]_
CT 4 verify:_[M-h][e8]__[M-}][fd]__[M-d][e4]__[M-^F][86]__[m][6d]__[M-G][
   c7]_
```

```
We have Verification also.
Limits: neednodes=128:ppn=8,nodes=128:ppn=8,walltime=04:00:00
Resources:     cput=1211:11:44,mem=209205236kb,vmem=394828684kb,walltime
   =01:11:27
```

Job ID 46040575

```
The number of booked cores is 1024 (1 Parent and 1023 children)
plaintext is: PTPTPT
Random key: (multiple) and it is used as the start point of the first
   chain.
After encrypting the plaintext T times (1048576 = 2^20, ) the chain end
   point becomes : (multiple)
Ciphertext Challenge: (multiple)
CT decrypts back to: (multiple)
output log: (multiple)
Output size (printf overhead): about 3170 lines, 323422 characters.
error log: (multiple)
Job log: (multiple)
source code file:(multiple)
job submission script:(multiple)
object file:(multiple)
Cores: 1024
usableCores: 1023
PID: (multiple)
Date: (multiple)
M= 1,072,693,248 = 2^30 records
C= 1024
W= 4 hours
T= 4096 = 2^12 iterations
B= 48
E= 2
H= 0
L= 0
S= 1
R= GPC
notes=VLowD4096t1024c1072693248m4hrs_bestCvrg0
```

Observation: Job 46040575 found the Key but also found 3 false positive leads. The three false positive leads are unique.

```
VLowD4096t1024c1072693248m4hrs_bestCvrg0.o46040575
46040575
Lead at t=1191 core=653, m=1036176, key1stB_^K_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[M-O][cf]__[^D][04]__[^N][0e]__[M-~][fe]__[|][7c]__[M-^Y][99]_
Recovered Key(EP-1):_[^K][0b]__[M-+][ab]__[M-V][d6]__[(][28]__[N][4e]__[
   ][20]_
Chain SP:_[^U][15]__[M-^U][95]__[M-9][b9]__[M-P][d0]__[M-X][d8]__[,][2c]_
Orig CT:_[5][35]__[M-&][a6]__[^W][17]__[M-I][c9]__[X][58]__["][22]_
Orig Key:_[t][74]__[4][34]__[b][62]__[V][56]__[r][72]__[y][79]_
CT 4 verify:_[M-i][e9]__[+][2b]__[M-_][df]__[^B][02]__[M-^\][9c]__[M-_][df
   ]_

Lead at t=1097 core=657, m=563376, key1stB_M-g_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[^^][1e]__[{][7b]__[M-^[][9b]__[H][48]__[M-^_][9f]__[^Z][1a]_
Recovered Key(EP-1):_[M-g][e7]__[M-J][ca]__[4][34]__[v][76]__[M-R][d2]__[^
   _][1f]_
Chain SP:_[M-}][fd]__[M-^_][9f]__[u][75]__[‘][60]__[{][7b]__[ ][20]_
Orig CT:_[5][35]__[M-&][a6]__[^W][17]__[M-I][c9]__[X][58]__["][22]_
Orig Key:_[t][74]__[4][34]__[b][62]__[V][56]__[r][72]__[y][79]_
CT 4 verify:_[5][35]__[M-^^][9e]__[%][25]__[M-|][fc]__[M-/][af]__[M-e][e5]
   _

Lead at t=353 core=728, m=1211874, key1stB_B_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[!][21]__[Q][51]__[^P][10]__[p][70]__[’][27]__[s][73]_
Recovered Key(EP-1):_[B][42]__[0][30]__[M-a][e1]__[6][36]__[>][3e]__[$
   ][24]_
Chain SP:_[^C][03]__[M-^V][96]__[0][30]__[M-2][b2]__[^Q][11]__[M-v][f6]_
Orig CT:_[5][35]__[M-&][a6]__[^W][17]__[M-I][c9]__[X][58]__["][22]_
Orig Key:_[t][74]__[4][34]__[b][62]__[V][56]__[r][72]__[y][79]_
CT 4 verify:_[M-^Z][9a]__[?][3f]__[M-+][ab]__[y][79]__[^^][1e]__[M-C][c3]_

Lead at t=1 core=1, m=0, key1stB_t_
```

```
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[M-2][b2]__[M-V][d6]__[M-B][c2]__[g][67]__[M-0][b0]__[N][4e]_
Recovered Key(EP-1):_[t][74]__[4][34]__[b][62]__[V][56]__[r][72]__[y][79]_
Chain SP:_[t][74]__[4][34]__[b][62]__[V][56]__[r][72]__[y][79]_
Orig CT:_[5][35]__[M-&][a6]__[^W][17]__[M-I][c9]__[X][58]__["][22]_
Orig Key:_[t][74]__[4][34]__[b][62]__[V][56]__[r][72]__[y][79]_
CT 4 verify:_[5][35]__[M-&][a6]__[^W][17]__[M-I][c9]__[X][58]__["][22]_
We have Verification also.

Limits: nednodes=128:ppn=8,nodes=128:ppn=8,walltime=04:00:00
Resources:     cput=1211:14:13,mem=209177124kb,vmem=394803344kb,walltime
   =01:11:29
```

Job ID 46040590

```
The number of booked cores is 1024 (1 Parent and 1023 children)
plaintext is: PTPTPT
Random key: (multiple) and it is used as the start point of the first
   chain.
After encrypting the plaintext T times (1048576 = 2^20, ) the chain end
   point becomes : (multiple)
Ciphertext Challenge: (multiple)
CT decrypts back to: (multiple)
output log: (multiple)
Output size (printf overhead): about 3170 lines, 323422 characters.
error log: (multiple)
Job log: (multiple)
source code file:(multiple)
job submission script:(multiple)
object file:(multiple)
Cores: 1024
usableCores: 1023
PID: (multiple) ....
Date: (multiple)
M= 1,072,693,248 = 2^30 records
C= 1024
W= 4 hours
```

```
T= 4096 = 2^12 iterations
B= 48
E= 2
H= 0
L= 0
S= 1
R= GPC
notes=VLowD4096t1024c1072693248m4hrs_bestCvrg0
```

Observation: Job 46040590 found the Key but also found 9 false positive leads. Two pairs had the same values: 0x031cfabd2efd and 0xb8e6f7103b14, one value was lost in the extraction but is recoverable, and the4 had unique values.

```
VLowD4096t1024c1072693248m8hrs_bestCvrg0.o46040590
46040590
Lead at t=1102 core=735, m=1026024, key1stB_^C_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[^?][7f]__[o][6f]__[^W][17]__[v][76]__[M-A][c1]__[M-:][ba]_
Recovered Key(EP-1):_[^C][03]__[^\][1c]__[M-z][fa]__[M-=][bd]__[.][2e]__[M
   -}][fd]_
Chain SP:_[^O][0f]__[M-9][b9]__[,][2c]__[m][6d]__[M-?][bf]__[}][7d]_
Orig CT:_[M-k][eb]__[^E][05]__[1][31]__[n][6e]__[M-"][a2]__[^Q][11]_
Orig Key:_[f][66]__[p][70]__[t][74]__[3][33]__[l][6c]__[q][71]_
CT 4 verify:_[^][5e]__[M-^Y][99]__[M-_][df]__[^V][16]__[8][38]__[A][41]_

Lead at t=973 core=563, m=1777236, key1stB_M-8_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[M-O][cf]__[M-s][f3]__[V][56]__[M-M][cd]__[M-^W][97]__[M-$][a4]_
Recovered Key(EP-1):_[M-8][b8]__[M-f][e6]__[M-w][f7]__[^P][10]__[;][3b]__
   [^T][14]_
Chain SP:_[,][2c]__[ ][09]__[^W][17]__[P][50]__[S][53]__[M-^G][87]_
Orig CT:_[M-k][eb]__[^E][05]__[1][31]__[n][6e]__[M-"][a2]__[^Q][11]_
Orig Key:_[f][66]__[p][70]__[t][74]__[3][33]__[l][6c]__[q][71]_
CT 4 verify:_[M-9][b9]__[M-^M][8d]__[M-^P][90]__[M-z][fa]__[{][7b]__[M-v][
   f6]_

Lead at t=1215 core=849, m=1274820, key1stB_M-^K_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
```

```
Found CT:_[M-J][ca]__[Y][59]__[
[Record truncated due to special characters.]

Lead at t=816 core=225, m=1277598, key1stB_M-W_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[^@][00]__[M-t][f4]__[M-7][b7]__[M-U][d5]__[M-"][a2]__[M-^W][97]
   _
Recovered Key(EP-1):_[M-W][d7]__[^][5e]__[z][7a]__[M-?][bf]__["][22]__[M-s
   ][f3]_
Chain SP:_[M-s][f3]__[h][68]__[G][47]__[M-^I][89]__[M-m][ed]__[^B][02]_
Orig CT:_[M-k][eb]__[^E][05]__[1][31]__[n][6e]__[M-"][a2]__[^Q][11]_
Orig Key:_[f][66]__[p][70]__[t][74]__[3][33]__[l][6c]__[q][71]_
CT 4 verify:_[^C][03]__[M-^M][8d]__[M-$][a4]__[M-u][f5]__[v][76]__[M-~][fe
   ]_

Lead at t=620 core=56, m=1130010, key1stB_‘_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[M-9][b9]__[~][7e]__[M-_][df]__[|][7c]__[n][6e]__[M-l][ec]_
Recovered Key(EP-1):_[‘][60]__[M-^\][9c]__[M-^E][85]__[M-g][e7]__[^L][0c]
   __[_][5f]_
Chain SP:_[^][5e]__[a][61]__[e][65]__[M-?][bf]__[N][4e]__[)][29]_
Orig CT:_[M-k][eb]__[^E][05]__[1][31]__[n][6e]__[M-"][a2]__[^Q][11]_
Orig Key:_[f][66]__[p][70]__[t][74]__[3][33]__[l][6c]__[q][71]_
CT 4 verify:_[S][53]__[M-~][fe]__[^A][01]__[0][30]__[$][24]__[M-%][a5]_

Lead at t=635 core=429, m=140364, key1stB_!_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[M-X][d8]__[f][66]__[M-^][de]__[C][43]__[M-’][a7]__[M-b][e2]_
Recovered Key(EP-1):_[!][21]__[Z][5a]__[2][32]__[M-^Y][99]__[)][29]__[M
   -.][ae]_
Chain SP:_[g][67]__[t][74]__[M-i][e9]__[v][76]__[6][36]__[F][46]_
Orig CT:_[M-k][eb]__[^E][05]__[1][31]__[n][6e]__[M-"][a2]__[^Q][11]_
Orig Key:_[f][66]__[p][70]__[t][74]__[3][33]__[l][6c]__[q][71]_
CT 4 verify:_[%][25]__[M-^H][88]__[^R][12]__[M][4d]__[M-$][a4]__[M-x][f8]_

Lead at t=1221 core=171, m=654936, key1stB_M-8_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[^_][1f]__[a][61]__[M-^O][8f]__[M-2][b2]__[M-g][e7]__[Y][59]_
```

```
Recovered Key(EP-1):_[M-8][b8]__[M-f][e6]__[M-w][f7]__[^P][10]__[;][3b]__
   [^T][14]_
Chain SP:_[M-^W][97]__[M-^F][86]__[A][41]__[M-A][c1]__[M-^J][8a]__[M-)][a9
   ]_
Orig CT:_[M-k][eb]__[^E][05]__[1][31]__[n][6e]__[M-"][a2]__[^Q][11]_
Orig Key:_[f][66]__[p][70]__[t][74]__[3][33]__[l][6c]__[q][71]_
CT 4 verify:_[M-9][b9]__[M-^M][8d]__[M-^P][90]__[M-z][fa]__[{][7b]__[M-v][
   f6]_

Lead at t=591 core=175, m=1235484, key1stB_^C_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[M-A][c1]__[M-~][fe]__["][22]__[M-u][f5]__[p][70]__[M-+][ab]_
Recovered Key(EP-1):_[^C][03]__[^\][1c]__[M-z][fa]__[M-=][bd]__[.][2e]__[M
   -}][fd]_
Chain SP:_[h][68]__[^C][03]__[M-t][f4]__[M-r][f2]__[M-9][b9]__[x][78]_
Orig CT:_[M-k][eb]__[^E][05]__[1][31]__[n][6e]__[M-"][a2]__[^Q][11]_
Orig Key:_[f][66]__[p][70]__[t][74]__[3][33]__[l][6c]__[q][71]_
CT 4 verify:_[^][5e]__[M-^Y][99]__[M-_][df]__[^V][16]__[8][38]__[A][41]_

Lead at t=162 core=720, m=421200, key1stB_M-"_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[M-1][b1]__[>][3e]__[5][35]__[M-=][bd]__[0][30]__[c][63]_
Recovered Key(EP-1):_[M-"][a2]__[^E][05]__[M-?][bf]__[ ][20]__[M-/][af]__[
   s][73]_
Chain SP:_[^?][7f]__[o][6f]__[M-{][fb]__[M-)][a9]__[<][3c]__[0][30]_
Orig CT:_[M-k][eb]__[^E][05]__[1][31]__[n][6e]__[M-"][a2]__[^Q][11]_
Orig Key:_[f][66]__[p][70]__[t][74]__[3][33]__[l][6c]__[q][71]_
CT 4 verify:_[M-^F][86]__[C][43]__[H][48]__[=][3d]__[^T][14]__[h][68]_

Lead at t=1 core=1, m=0, key1stB_f_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[4][34]__[i][69]__[M-Z][da]__[M-J][ca]__[^[][1b]__[M-^^][9e]_
Recovered Key(EP-1):_[f][66]__[p][70]__[t][74]__[3][33]__[l][6c]__[q][71]_
Chain SP:_[f][66]__[p][70]__[t][74]__[3][33]__[l][6c]__[q][71]_
Orig CT:_[M-k][eb]__[^E][05]__[1][31]__[n][6e]__[M-"][a2]__[^Q][11]_
Orig Key:_[f][66]__[p][70]__[t][74]__[3][33]__[l][6c]__[q][71]_
CT 4 verify:_[M-k][eb]__[^E][05]__[1][31]__[n][6e]__[M-"][a2]__[^Q][11]_
We have Verification also.
```

```
Limits: neednodes=128:ppn=8,nodes=128:ppn=8,walltime=08:00:00
Resources:     cput=1211:06:43,mem=209311840kb,vmem=394929052kb,walltime
   =01:11:36
```

Job ID 46040591

```
The number of booked cores is 1024 (1 Parent and 1023 children)
plaintext is: PTPTPT
Random key: (multiple) and it is used as the start point of the first
   chain.
After encrypting the plaintext T times (1048576 = 2^20, ) the chain end
   point becomes : (multiple)
Ciphertext Challenge: (multiple)
CT decrypts back to: (multiple)
output log: (multiple)
Output size (printf overhead): about 3170 lines, 323422 characters.
error log: (multiple)
Job log: (multiple)
source code file:(multiple)
job submission script:(multiple)
object file:(multiple)
Cores: 1024
usableCores: 1023
PID: (multiple) ....
Date: (multiple)
M= 1,072,693,248 = 2^30 records
C= 1024
W= 4 hours
T= 4096 = 2^12 iterations
B= 48
E= 2
H= 0
L= 0
S= 1
R= GPC
notes=VLowD4096t1024c1072693248m4hrs_bestCvrg0
```

Observation: Job 46040591 found the Key but also found 18 false positive leads. One false positive value (0xc715ed95a111) repeats 6 times in 6 different chains. One repeats 3 times, and two repeats twice. five more are unique. Since this is a significantly larger set of false positives we can compare t, c, and m to see if there is any relations. Figure A.2 shows that only one value repeats c=42 which means that core 42 yielded 2 false positives at 2 different chains in the same table, chain 1026006 and chain 1639710. Each happened at a different iteration t, respectively, t=599, and t=1861.

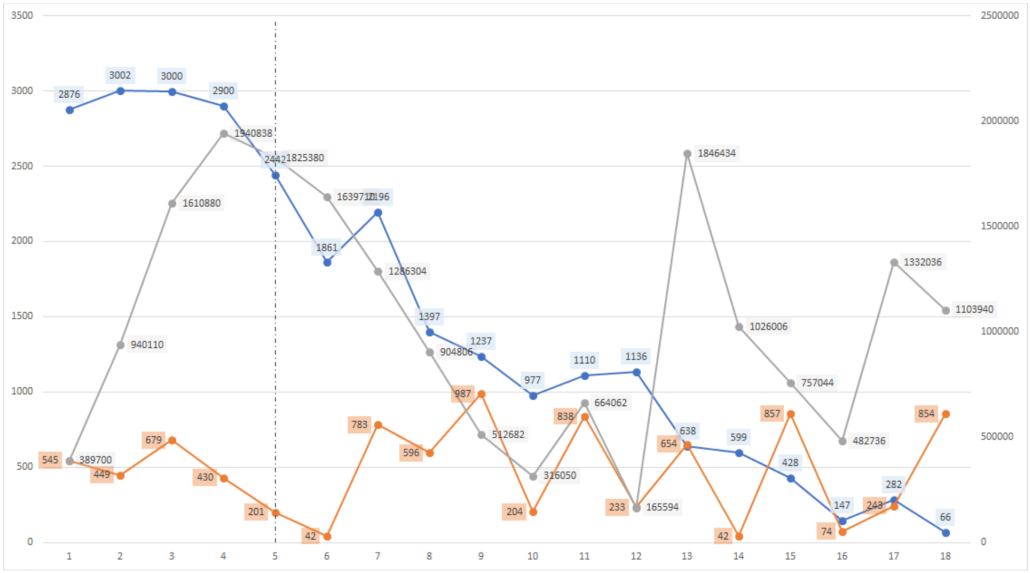

Figure A.2: False positive plot for job 46040591

```
VLowD4096t1024c1072693248m8hrs_bestCvrg0.o46040591
46040591

Lead at t=2876 core=545, m=389700, key1stB_M-G_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[w][77]__[E][45]__[L][4c]__[M-^Z][9a]__[M-k][eb]__[7][37]_
Recovered Key(EP-1):_[M-G][c7]__[^U][15]__[M-m][ed]__[M-^U][95]__[M-!][a1]
    __[^Q][11]_
Chain SP:_[6][36]__[B][42]__[d][64]__[
[String truncated while extracting]

Lead at t=3002 core=449, m=940110, key1stB_M-G_
```

```
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[M-^\][9c]__[U][55]__[M-A][c1]__[^X][18]__[3][33]__[M-^X][98]_
Recovered Key(EP-1):_[M-G][c7]__[^U][15]__[M-m][ed]__[M-^U][95]__[M-!][a1]
    __[^Q][11]_
Chain SP:_()][29]__[M-N][ce]__[D][44]__[M-^G][87]__[|][7c]__[M-^J][8a]_
Orig CT:_[M-_][df]__[e][65]__[^S][13]__[M-B][c2]__[M-/][af]__[M-p][f0]_
Orig Key:_[i][69]__[Y][59]__[I][49]__[T][54]__[2][32]__[o][6f]_
CT 4 verify:_[M-8][b8]__[(][28]__[^E][05]__[%][25]__[M- ][a0]__[C][43]_

Lead at t=3000 core=679, m=1610880, key1stB_M-G_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[@][40]__[M-^]][9d]__[&][26]__[^E][05]__[^X][18]__[W][57]_
Recovered Key(EP-1):_[M-G][c7]__[^U][15]__[M-m][ed]__[M-^U][95]__[M-!][a1]
    __[^Q][11]_
Chain SP:_[+][2b]__[@][40]__[4][34]__[^C][03]__[r][72]__[M-^?][ff]_
Orig CT:_[M-_][df]__[e][65]__[^S][13]__[M-B][c2]__[M-/][af]__[M-p][f0]_
Orig Key:_[i][69]__[Y][59]__[I][49]__[T][54]__[2][32]__[o][6f]_
CT 4 verify:_[M-8][b8]__[(][28]__[^E][05]__[%][25]__[M- ][a0]__[C][43]_

Lead at t=2900 core=430, m=1940838, key1stB_M-G_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[N][4e]__[M-l][ec]__[x][78]__[^M][0d]__[K][4b]__[M-B][c2]_
Recovered Key(EP-1):_[M-G][c7]__[^U][15]__[M-m][ed]__[M-^U][95]__[M-!][a1]
    __[^Q][11]_
Chain SP:_[M-Q][d1]__[k][6b]__[M-:][ba]__[M-^B][82]__[^^][1e]__[M-^T][94]_
Orig CT:_[M-_][df]__[e][65]__[^S][13]__[M-B][c2]__[M-/][af]__[M-p][f0]_
Orig Key:_[i][69]__[Y][59]__[I][49]__[T][54]__[2][32]__[o][6f]_
CT 4 verify:_[M-8][b8]__[(][28]__[^E][05]__[%][25]__[M- ][a0]__[C][43]_

Lead at t=2442 core=201, m=1825380, key1stB_M-G_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[M-^G][87]__[M-^F][86]__[1][31]__[M-z][fa]__[M-^C][83]__[S][53]_
Recovered Key(EP-1):_[M-G][c7]__[^U][15]__[M-m][ed]__[M-^U][95]__[M-!][a1]
    __[^Q][11]_
Chain SP:_[M-w][f7]__[^W][17]__[5][35]__[M-h][e8]__[M-S][d3]__[M-^?][ff]_
Orig CT:_[M-_][df]__[e][65]__[^S][13]__[M-B][c2]__[M-/][af]__[M-p][f0]_
Orig Key:_[i][69]__[Y][59]__[I][49]__[T][54]__[2][32]__[o][6f]_
CT 4 verify:_[M-8][b8]__[(][28]__[^E][05]__[%][25]__[M- ][a0]__[C][43]_
```

```
Lead at t=1861 core=42, m=1639710, key1stB_^A_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[M-F][c6]__[G][47]__[M-?][bf]__[^V][16]__[G][47]__[M-^N][8e]_
Recovered Key(EP-1):_[^A][01]__[]][5d]__[n][6e]__[l][6c]__[^P][10]__[^A
   ][01]_
Chain SP:_[M][4d]__[H][48]__[M-[][db]__[%][25]__[6][36]__[M-*][aa]_
Orig CT:_[M-_][df]__[e][65]__[^S][13]__[M-B][c2]__[M-/][af]__[M-p][f0]_
Orig Key:_[i][69]__[Y][59]__[I][49]__[T][54]__[2][32]__[o][6f]_
CT 4 verify:_[^L][0c]__[7][37]__[M-v][f6]__[M-X][d8]__[M-^?][ff]__[M-L][cc
   ]_

Lead at t=2196 core=783, m=1286304, key1stB_M-^F_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[?][3f]__[5][35]__[M-~][fe]__[M-^F][86]__[^E][05]__[T][54]_
Recovered Key(EP-1):_[M-^F][86]__[^@][00]__[^S][13]__[^T][14]__[i][69]__[M
   -^O][8f]_
Chain SP:_[M-;][bb]__[^?][7f]__[k][6b]__[M-^Z][9a]__[M-0][b0]__[M- ][a0]_
Orig CT:_[M-_][df]__[e][65]__[^S][13]__[M-B][c2]__[M-/][af]__[M-p][f0]_
Orig Key:_[i][69]__[Y][59]__[I][49]__[T][54]__[2][32]__[o][6f]_
CT 4 verify:_[M-c][e3]__[M-t][f4]__[]][5d]__[M-^J][8a]__[M-.][ae]__[X][58]
   -

Lead at t=1397 core=596, m=904806, key1stB_M-^F_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[M-M][cd]__[M-Y][d9]__[M-W][d7]__[^Q][11]__[q][71]__[l][6c]_
Recovered Key(EP-1):_[M-^F][86]__[^@][00]__[^S][13]__[^T][14]__[i][69]__[M
   -^O][8f]_
Chain SP:_[M-^@][80]__[^T][14]__[u][75]__[M-^I][89]__[M-z][fa]__[a][61]_
Orig CT:_[M-_][df]__[e][65]__[^S][13]__[M-B][c2]__[M-/][af]__[M-p][f0]_
Orig Key:_[i][69]__[Y][59]__[I][49]__[T][54]__[2][32]__[o][6f]_
CT 4 verify:_[M-c][e3]__[M-t][f4]__[]][5d]__[M-^J][8a]__[M-.][ae]__[X][58]
   -

Lead at t=1237 core=987, m=512682, key1stB_M-G_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[M-B][c2]__["][22]__[M-|][fc]__[M-^J][8a]__[M-^R][92]__[^_][1f]_
Recovered Key(EP-1):_[M-G][c7]__[^U][15]__[M-m][ed]__[M-^U][95]__[M-!][a1]
```

```
   __[^Q][11]_
Chain SP:_[M-|][fc]__[^V][16]__[Z][5a]__[M-?][bf]__[T][54]__[M-K][cb]_
Orig CT:_[M-_][df]__[e][65]__[^S][13]__[M-B][c2]__[M-/][af]__[M-p][f0]_
Orig Key:_[i][69]__[Y][59]__[I][49]__[T][54]__[2][32]__[o][6f]_
CT 4 verify:_[M-8][b8]__[(][28]__[^E][05]__[%][25]__[M- ][a0]__[C][43]_

Lead at t=977 core=204, m=316050, key1stB_M-^__
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[M-H][c8]__[M-m][ed]__[C][43]__[M-‘][e0]__[M-F][c6]__[M-0][b0]_
Recovered Key(EP-1):_[M-^_][9f]__[M-X][d8]__[^V][16]__[y][79]__[M-Q][d1]__
   [M-^H][88]_
Chain SP:_[M-L][cc]__[M-.][ae]__[M-v][f6]__[^[][1b]__[M-]][dd]__[M-)][a9]_
Orig CT:_[M-_][df]__[e][65]__[^S][13]__[M-B][c2]__[M-/][af]__[M-p][f0]_
Orig Key:_[i][69]__[Y][59]__[I][49]__[T][54]__[2][32]__[o][6f]_
CT 4 verify:_[M-Y][d9]__[M-*][aa]__[M-n][ee]__[r][72]__[)][29]__[^V][16]_

Lead at t=1110 core=838, m=664062, key1stB_M-^__
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[M-<][bc]__[^Z][1a]__[M-p][f0]__[=][3d]__[M-V][d6]__[ ][09]_
Recovered Key(EP-1):_[M-^_][9f]__[M-X][d8]__[^V][16]__[y][79]__[M-Q][d1]__
   [M-^H][88]_
Chain SP:_[^T][14]__[M-^D][84]__[^K][0b]__[M-f][e6]__[M-G][c7]__[Z][5a]_
Orig CT:_[M-_][df]__[e][65]__[^S][13]__[M-B][c2]__[M-/][af]__[M-p][f0]_
Orig Key:_[i][69]__[Y][59]__[I][49]__[T][54]__[2][32]__[o][6f]_
CT 4 verify:_[M-Y][d9]__[M-*][aa]__[M-n][ee]__[r][72]__[)][29]__[^V][16]_

Lead at t=1136 core=233, m=165594, key1stB_E_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[M-i][e9]__[2][32]__[T][54]__[^@][00]__[_][5f]__[M-J][ca]_
Recovered Key(EP-1):_[E][45]__[^X][18]__[M-^T][94]__[^N][0e]__[M-^Y][99]__
   [M-L][cc]_
Chain SP:_[M-‘][e0]__[M-z][fa]__[M-z][fa]__[M-u][f5]__[M-v][f6]__[M-i][e9]
   _
Orig CT:_[M-_][df]__[e][65]__[^S][13]__[M-B][c2]__[M-/][af]__[M-p][f0]_
Orig Key:_[i][69]__[Y][59]__[I][49]__[T][54]__[2][32]__[o][6f]_
CT 4 verify:_[M-{][fb]__[M-s][f3]__[u][75]__[^B][02]__[2][32]__[M-1][b1]_

Lead at t=638 core=654, m=1846434, key1stB_M-^F_
```

```
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[:][3a]__[M-l][ec]__[?][3f]__["][22]__[n][6e]__[M-^R][92]_
Recovered Key(EP-1):_[M-^F][86]__[^@][00]__[^S][13]__[^T][14]__[i][69]__[M
   -^O][8f]_
Chain SP:_[M-}][fd]__[M-^L][8c]__[M-^R][92]__[M-z][fa]__[M-t][f4]__[M-^J
   ][8a]_
Orig CT:_[M-_][df]__[e][65]__[^S][13]__[M-B][c2]__[M-/][af]__[M-p][f0]_
Orig Key:_[i][69]__[Y][59]__[I][49]__[T][54]__[2][32]__[o][6f]_
CT 4 verify:_[M-c][e3]__[M-t][f4]__[]][5d]__[M-^J][8a]__[M-.][ae]__[X][58]
   _

Lead at t=599 core=42, m=1026006, key1stB_w_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[M-F][c6]__[G][47]__[M-?][bf]__[^V][16]__[G][47]__[M-^N][8e]_
Recovered Key(EP-1):_[w][77]__[^@][00]__[]][5d]__[z][7a]__[M-V][d6]__
   [^^][1e]_
Chain SP:_[M][4d]__[H][48]__[M-[][db]__[%][25]__[6][36]__[M-*][aa]_
Orig CT:_[M-_][df]__[e][65]__[^S][13]__[M-B][c2]__[M-/][af]__[M-p][f0]_
Orig Key:_[i][69]__[Y][59]__[I][49]__[T][54]__[2][32]__[o][6f]_
CT 4 verify:_[@][40]__[M-V][d6]__[M-^?][ff]__[y][79]__[s][73]__[+][2b]_

Lead at t=428 core=857, m=757044, key1stB_E_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[M-E][c5]__[^][5e]__[I][49]__[\][5c]__[3][33]__[M-<][bc]_
Recovered Key(EP-1):_[E][45]__[^X][18]__[M-^T][94]__[^N][0e]__[M-^Y][99]__
   [M-L][cc]_
Chain SP:_[I][49]__[M-^N][8e]__[]][5d]__[#][23]__[e][65]__[V][56]_
Orig CT:_[M-_][df]__[e][65]__[^S][13]__[M-B][c2]__[M-/][af]__[M-p][f0]_
Orig Key:_[i][69]__[Y][59]__[I][49]__[T][54]__[2][32]__[o][6f]_
CT 4 verify:_[M-{][fb]__[M-s][f3]__[u][75]__[^B][02]__[2][32]__[M-1][b1]_

Lead at t=147 core=74, m=482736, key1stB_M-%_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[^Z][1a]__[M-^[][9b]__[M-E][c5]__[M-v][f6]__[M-^J][8a]__[&][26]_
Recovered Key(EP-1):_[M-%][a5]__[{][7b]__[k][6b]__[M-]][dd]__[M-|][fc]__[M
   -^K][8b]_
Chain SP:_[M-$][a4]__[x][78]__[M-^V][96]__[?][3f]__["][22]__[^H][08]_
Orig CT:_[M-_][df]__[e][65]__[^S][13]__[M-B][c2]__[M-/][af]__[M-p][f0]_
```

```
Orig Key:_[i][69]__[Y][59]__[I][49]__[T][54]__[2][32]__[o][6f]_
CT 4 verify:_[^T][14]__[‘][60]__[)][29]__[M-^]][9d]__[7][37]__[&][26]_

Lead at t=282 core=243, m=1332036, key1stB_M-:_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[,][2c]__[8][38]__[M-^X][98]__[|][7c]__[n][6e]__[^P][10]_
Recovered Key(EP-1):_[M-:][ba]__[^B][02]__[>][3e]__[M-i][e9]__[K][4b]__[v
   ][76]_
Chain SP:_[M-)][a9]__[(][28]__[M-e][e5]__[^H][08]__[M-!][a1]__[M-^K][8b]_
Orig CT:_[M-_][df]__[e][65]__[^S][13]__[M-B][c2]__[M-/][af]__[M-p][f0]_
Orig Key:_[i][69]__[Y][59]__[I][49]__[T][54]__[2][32]__[o][6f]_
CT 4 verify:_[M-9][b9]__[S][53]__[]][5d]__[Y][59]__[M-^O][8f]__[b][62]_

Lead at t=1 core=1, m=0, key1stB_i_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[M-s][f3]__[d][64]__[M-L][cc]__[M-V][d6]__[9][39]__[7][37]_
Recovered Key(EP-1):_[i][69]__[Y][59]__[I][49]__[T][54]__[2][32]__[o][6f]_
Chain SP:_[i][69]__[Y][59]__[I][49]__[T][54]__[2][32]__[o][6f]_
Orig CT:_[M-_][df]__[e][65]__[^S][13]__[M-B][c2]__[M-/][af]__[M-p][f0]_
Orig Key:_[i][69]__[Y][59]__[I][49]__[T][54]__[2][32]__[o][6f]_
CT 4 verify:_[M-_][df]__[e][65]__[^S][13]__[M-B][c2]__[M-/][af]__[M-p][f0]
   _
We have Verification also.


Lead at t=66 core=854, m=1103940, key1stB_X_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[M-v][f6]__[M-N][ce]__[M-s][f3]__[1][31]__[M-S][d3]__[M-U][d5]_
Recovered Key(EP-1):_[X][58]__[&][26]__[M-^V][96]__[}][7d]__[M-h][e8]__[q
   ][71]_
Chain SP:_[M-/][af]__[M-^L][8c]__[M-T][d4]__[M-^P][90]__[}][7d]__[M-p][f0]
   _
Orig CT:_[M-_][df]__[e][65]__[^S][13]__[M-B][c2]__[M-/][af]__[M-p][f0]_
Orig Key:_[i][69]__[Y][59]__[I][49]__[T][54]__[2][32]__[o][6f]_
CT 4 verify:_[^E][05]__[^][5e]__[M-~][fe]__[M-D][c4]__[o][6f]__[M-^L][8c]_

Limits: nednodes=128:ppn=8,nodes=128:ppn=8,walltime=08:00:00
Resources:     cput=1211:09:26,mem=209298680kb,vmem=394924508kb,walltime
```

```
   =01:11:33
```

Job ID 46040592

```
The number of booked cores is 1024 (1 Parent and 1023 children)
plaintext is: PTPTPT
Random key: (multiple) and it is used as the start point of the first
   chain.
After encrypting the plaintext T times (1048576 = 2^20, ) the chain end
   point becomes : (multiple)
Ciphertext Challenge: (multiple)
CT decrypts back to: (multiple)
output log: (multiple)
Output size (printf overhead): about 3170 lines, 323422 characters.
error log: (multiple)
Job log: (multiple)
source code file:(multiple)
job submission script:(multiple)
object file:(multiple)
Cores: 1024
usableCores: 1023
PID: (multiple) ....
Date: (multiple)
M= 1,072,693,248 = 2^30 records
C= 1024
W= 4 hours
T= 4096 = 2^12 iterations
B= 48
E= 2
H= 0
L= 0
S= 1
R= GPC
VLowD4096t1024c1072693248m8hrs_bestCvrg0
```

Observation: Job 46040592 found the Key but also found 16 false positive leads. One false positive value (0x356d35bca3ef) repeats 12 times in 12 different chains. The remaining 4

false positives are unique. This is also a significantly large set of false positives. We can compare t, c, and m to see if there is any relations. Figure A.3 shows that there is no overlap.

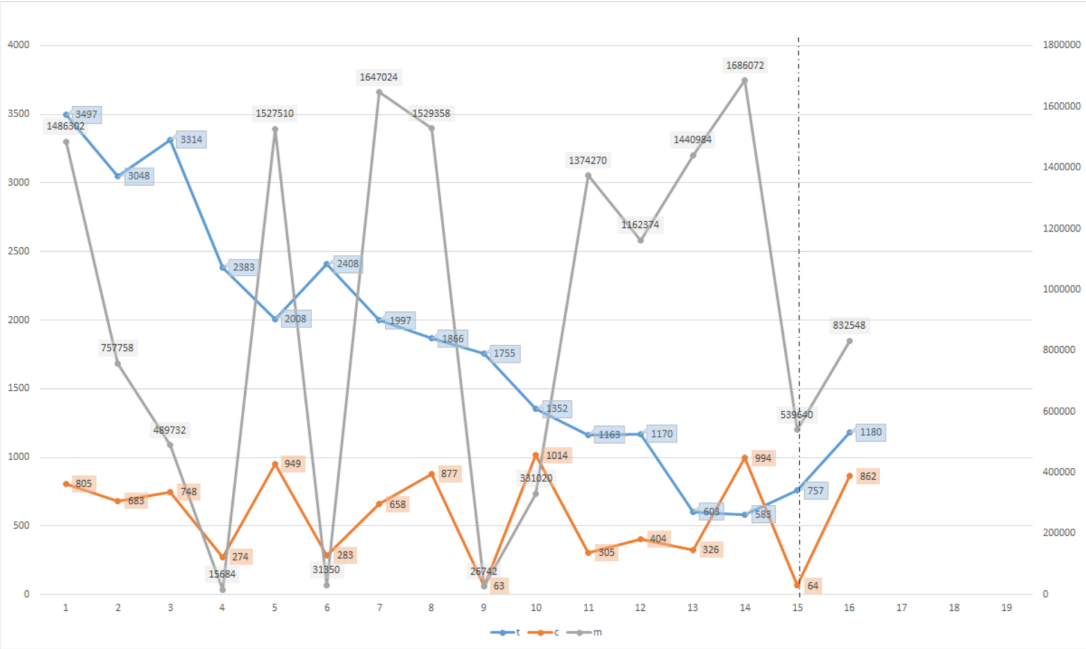

Figure A.3: False positive plot for job 46040592

```
VLowD4096t1024c1072693248m8hrs_bestCvrg0.o46040592
46040592
Lead at t=3497 core=805, m=1486302, key1stB_5_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[?][3f]__[M-;][bb]__[a][61]__[M-^G][87]__[M-f][e6]__[M-C][c3]_
Recovered Key(EP-1):_[5][35]__[m][6d]__[5][35]__[M-<][bc]__[M-#][a3]__[M-o
    ][ef]_
Chain SP:_[M-^T][94]__[U][55]__[
[Record truncated due to special characters.]

Lead at t=3048 core=683, m=757758, key1stB_5_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[~][7e]__[M-w][f7]__[M-o][ef]__[M-e][e5]__[Z][5a]__[Z][5a]_
```

```
Recovered Key(EP-1):_[5][35]__[m][6d]__[5][35]__[M-<][bc]__[M-#][a3]__[M-o
   ][ef]_
Chain SP:_[|][7c]__[y][79]__[M-Y][d9]__[p][70]__[^N][0e]__[M-^V][96]_
Orig CT:_[a][61]__[M-!][a1]__[A][41]__[r][72]__[M-^][de]__[c][63]_
Orig Key:_[s][73]__[l][6c]__[m][6d]__[e][65]__[8][38]__[N][4e]_
CT 4 verify:_[M-^Z][9a]__[^U][15]__[^\][1c]__[M-^C][83]__[^D][04]__[^][5e]
   _

Lead at t=3314 core=748, m=489732, key1stB_5_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[6][36]__[f][66]__[^U][15]__[M-l][ec]__[M-^D][84]__[^_][1f]_
Recovered Key(EP-1):_[5][35]__[m][6d]__[5][35]__[M-<][bc]__[M-#][a3]__[M-o
   ][ef]_
Chain SP:_[*][2a]__[^A][01]__[V][56]__[M-G][c7]__[M-\][dc]__[2][32]_
Orig CT:_[a][61]__[M-!][a1]__[A][41]__[r][72]__[M-^][de]__[c][63]_
Orig Key:_[s][73]__[l][6c]__[m][6d]__[e][65]__[8][38]__[N][4e]_
CT 4 verify:_[M-^Z][9a]__[^U][15]__[^\][1c]__[M-^C][83]__[^D][04]__[^][5e]
   _

Lead at t=2383 core=274, m=15684, key1stB_5_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[M->][be]__[M-4][b4]__[M-^\][9c]__[M-"][a2]__[\][5c]__[M-G][c7]_
Recovered Key(EP-1):_[5][35]__[m][6d]__[5][35]__[M-<][bc]__[M-#][a3]__[M-o
   ][ef]_
Chain SP:_[>][3e]__[%][25]__[M-x][f8]__[M-n][ee]__[M-A][c1]__[@][40]_
Orig CT:_[a][61]__[M-!][a1]__[A][41]__[r][72]__[M-^][de]__[c][63]_
Orig Key:_[s][73]__[l][6c]__[m][6d]__[e][65]__[8][38]__[N][4e]_
CT 4 verify:_[M-^Z][9a]__[^U][15]__[^\][1c]__[M-^C][83]__[^D][04]__[^][5e]
   _

Lead at t=2008 core=949, m=1527510, key1stB_5_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[O][4f]__[M-V][d6]__[}][7d]__[^_][1f]__[^C][03]__[^L][0c]_
Recovered Key(EP-1):_[5][35]__[m][6d]__[5][35]__[M-<][bc]__[M-#][a3]__[M-o
   ][ef]_
Chain SP:_[F][46]__[M-^I][89]__[M-x][f8]__[M-~][fe]__[M-q][f1]__[M-Y][d9]_
Orig CT:_[a][61]__[M-!][a1]__[A][41]__[r][72]__[M-^][de]__[c][63]_
Orig Key:_[s][73]__[l][6c]__[m][6d]__[e][65]__[8][38]__[N][4e]_
```

```
CT 4 verify:_[M-^Z][9a]__[^U][15]__[^\][1c]__[M-^C][83]__[^D][04]__[^][5e]
   _

Lead at t=2408 core=283, m=31350, key1stB_5_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[^S][13]__[M-U][d5]__[5][35]__[M-Z][da]__[U][55]__[/][2f]_
Recovered Key(EP-1):_[5][35]__[m][6d]__[5][35]__[M-<][bc]__[M-#][a3]__[M-o
   ][ef]_
Chain SP:_[M-[][db]__[M-B][c2]__[I][49]__[B][42]__[E][45]__[G][47]_
Orig CT:_[a][61]__[M-!][a1]__[A][41]__[r][72]__[M-^][de]__[c][63]_
Orig Key:_[s][73]__[l][6c]__[m][6d]__[e][65]__[8][38]__[N][4e]_
CT 4 verify:_[M-^Z][9a]__[^U][15]__[^\][1c]__[M-^C][83]__[^D][04]__[^][5e]
   _

Lead at t=1997 core=658, m=1647024, key1stB_\_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[/][2f]__[M-'][a7]__[e][65]__[^T][14]__[ ][20]__[M-^X][98]_
Recovered Key(EP-1):_[\][5c]__[M-d][e4]__[Y][59]__[_][5f]__[{][7b]__[M-.][
   ae]_
Chain SP:_[5][35]__[M-~][fe]__[8][38]__[}][7d]__[M-u][f5]__[M-^@][80]_
Orig CT:_[a][61]__[M-!][a1]__[A][41]__[r][72]__[M-^][de]__[c][63]_
Orig Key:_[s][73]__[l][6c]__[m][6d]__[e][65]__[8][38]__[N][4e]_
CT 4 verify:_[M-^]][9d]__[M-c][e3]__[A][41]__[l][6c]__[>][3e]__[X][58]_

Lead at t=1866 core=877, m=1529358, key1stB_5_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[R][52]__[{][7b]__[M-j][ea]__[M-:][ba]__[6][36]__[M-:][ba]_
Recovered Key(EP-1):_[5][35]__[m][6d]__[5][35]__[M-<][bc]__[M-#][a3]__[M-o
   ][ef]_
Chain SP:_[6][36]__[M-(][a8]__[M-]][dd]__[^@][00]__[>][3e]__[M->][be]_
Orig CT:_[a][61]__[M-!][a1]__[A][41]__[r][72]__[M-^][de]__[c][63]_
Orig Key:_[s][73]__[l][6c]__[m][6d]__[e][65]__[8][38]__[N][4e]_
CT 4 verify:_[M-^Z][9a]__[^U][15]__[^\][1c]__[M-^C][83]__[^D][04]__[^][5e]
   _

Lead at t=1755 core=63, m=26742, key1stB_5_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[^K][0b]__[%][25]__[M-[][db]__[M-@][c0]__[M-k][eb]__[l][6c]_
```

```
Recovered Key(EP-1):_[5][35]__[m][6d]__[5][35]__[M-<][bc]__[M-#][a3]__[M-o
   ][ef]_
Chain SP:_[M-^Q][91]__[M-v][f6]__[M-C][c3]__[M-^Y][99]__[M-_][df]__[M-L][
   cc]_
Orig CT:_[a][61]__[M-!][a1]__[A][41]__[r][72]__[M-^][de]__[c][63]_
Orig Key:_[s][73]__[l][6c]__[m][6d]__[e][65]__[8][38]__[N][4e]_
CT 4 verify:_[M-^Z][9a]__[^U][15]__[^\][1c]__[M-^C][83]__[^D][04]__[^][5e]
   _

Lead at t=1352 core=1014, m=331020, key1stB_5_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[M-m][ed]__[M-^N][8e]__[M-=][bd]__[x][78]__[.][2e]__[b][62]_
Recovered Key(EP-1):_[5][35]__[m][6d]__[5][35]__[M-<][bc]__[M-#][a3]__[M-o
   ][ef]_
Chain SP:_[M-'][a7]__[v][76]__[u][75]__[b][62]__[$][24]__[M-^I][89]_
Orig CT:_[a][61]__[M-!][a1]__[A][41]__[r][72]__[M-^][de]__[c][63]_
Orig Key:_[s][73]__[l][6c]__[m][6d]__[e][65]__[8][38]__[N][4e]_
CT 4 verify:_[M-^Z][9a]__[^U][15]__[^\][1c]__[M-^C][83]__[^D][04]__[^][5e]
   _

Lead at t=1163 core=305, m=1374270, key1stB_5_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[M-y][f9]__[M-+][ab]__[M-R][d2]__[^Z][1a]__[M-^C][83]__[M-^M][8d
   ]_
Recovered Key(EP-1):_[5][35]__[m][6d]__[5][35]__[M-<][bc]__[M-#][a3]__[M-o
   ][ef]_
Chain SP:_[M-*][aa]__[M-i][e9]__[M-1][b1]__[o][6f]__[M-k][eb]__[M-8][b8]_
Orig CT:_[a][61]__[M-!][a1]__[A][41]__[r][72]__[M-^][de]__[c][63]_
Orig Key:_[s][73]__[l][6c]__[m][6d]__[e][65]__[8][38]__[N][4e]_
CT 4 verify:_[M-^Z][9a]__[^U][15]__[^\][1c]__[M-^C][83]__[^D][04]__[^][5e]
   _

Lead at t=1170 core=404, m=1162374, key1stB_5_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[a][61]__[M-`][e0]__[^X][18]__[M-^S][93]__[g][67]__[;][3b]_
Recovered Key(EP-1):_[5][35]__[m][6d]__[5][35]__[M-<][bc]__[M-#][a3]__[M-o
   ][ef]_
Chain SP:_[M-"][a2]__[M-G][c7]__[+][2b]__[.][2e]__[M-^D][84]__[M-'][a7]_
```

```
Orig CT:_[a][61]__[M-!][a1]__[A][41]__[r][72]__[M-^][de]__[c][63]_
Orig Key:_[s][73]__[l][6c]__[m][6d]__[e][65]__[8][38]__[N][4e]_
CT 4 verify:_[M-^Z][9a]__[^U][15]__[^\][1c]__[M-^C][83]__[^D][04]__[^][5e]
   _

Lead at t=603 core=326, m=1440984, key1stB_5_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[M-^][de]__[b][62]__[M-M][cd]__[/][2f]__[!][21]__[M-g][e7]_
Recovered Key(EP-1):_[5][35]__[M-]][dd]__[M-X][d8]__[M-B][c2]__[^Q][11]__[
   M-B][c2]_
Chain SP:_[6][36]__[M-^A][81]__[M-^O][8f]__[!][21]__[M-s][f3]__[,][2c]_
Orig CT:_[a][61]__[M-!][a1]__[A][41]__[r][72]__[M-^][de]__[c][63]_
Orig Key:_[s][73]__[l][6c]__[m][6d]__[e][65]__[8][38]__[N][4e]_
CT 4 verify:_[M-K][cb]__[F][46]__[^H][08]__[t][74]__[M-^O][8f]__[M-^A][81]
   _

Lead at t=583 core=994, m=1686072, key1stB_?_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[b][62]__[^L][0c]__[M-^L][8c]__[M-R][d2]__[M-#][a3]__[B][42]_
Recovered Key(EP-1):_[?][3f]__[M-%][a5]__[^Y][19]__[9][39]__[M-.][ae]__[c
   ][63]_
Chain SP:_[{][7b]__[M-T][d4]__[M-^R][92]__[M-^T][94]__[#][23]__[M-4][b4]_
Orig CT:_[a][61]__[M-!][a1]__[A][41]__[r][72]__[M-^][de]__[c][63]_
Orig Key:_[s][73]__[l][6c]__[m][6d]__[e][65]__[8][38]__[N][4e]_
CT 4 verify:_[^H][08]__[~][7e]__[^T][14]__[M-Y][d9]__[%][25]__[9][39]_

Lead at t=757 core=64, m=539640, key1stB_i_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[v][76]__[Z][5a]__[M-‘][e0]__[^R][12]__[4][34]__[A][41]_
Recovered Key(EP-1):_[i][69]__[e][65]__[,][2c]__[M-N][ce]__[^L][0c]__[M-c
   ][e3]_
Chain SP:_[M-2][b2]__[2][32]__[#][23]__[i][69]__[M-t][f4]__[M-\][dc]_
Orig CT:_[a][61]__[M-!][a1]__[A][41]__[r][72]__[M-^][de]__[c][63]_
Orig Key:_[s][73]__[l][6c]__[m][6d]__[e][65]__[8][38]__[N][4e]_
CT 4 verify:_[M-E][c5]__[^F][06]__[M-[][db]__[M-"][a2]__[M-^D][84]__[s
   ][73]_

Lead at t=1180 core=862, m=832548, key1stB_5_
```

```
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[^_][1f]__[p][70]__[R][52]__[M-.][ae]__[M-!][a1]__[c][63]_
Recovered Key(EP-1):_[5][35]__[m][6d]__[5][35]__[M-<][bc]__[M-#][a3]__[M-o
   ][ef]_
Chain SP:_[M-|][fc]__[M-m][ed]__[M-R][d2]__[^M][0d]__[b][62]__[M-6][b6]_
Orig CT:_[a][61]__[M-!][a1]__[A][41]__[r][72]__[M-^][de]__[c][63]_
Orig Key:_[s][73]__[l][6c]__[m][6d]__[e][65]__[8][38]__[N][4e]_
CT 4 verify:_[M-^Z][9a]__[^U][15]__[^\][1c]__[M-^C][83]__[^D][04]__[^][5e]
   _

Lead at t=1 core=1, m=0, key1stB_s_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[n][6e]__[F][46]__[|][7c]__[M-^X][98]__[^X][18]__[~][7e]_
Recovered Key(EP-1):_[s][73]__[l][6c]__[m][6d]__[e][65]__[8][38]__[N][4e]_
Chain SP:_[s][73]__[l][6c]__[m][6d]__[e][65]__[8][38]__[N][4e]_
Orig CT:_[a][61]__[M-!][a1]__[A][41]__[r][72]__[M-^][de]__[c][63]_
Orig Key:_[s][73]__[l][6c]__[m][6d]__[e][65]__[8][38]__[N][4e]_
CT 4 verify:_[a][61]__[M-!][a1]__[A][41]__[r][72]__[M-^][de]__[c][63]_
We have Verification also.

Limits: nednodes=128:ppn=8,nodes=128:ppn=8,walltime=08:00:00
Resources:     cput=1211:07:26,mem=209285908kb,vmem=394917648kb,walltime
   =01:11:27
```

Job ID 46040593

```
The number of booked cores is 1024 (1 Parent and 1023 children)
plaintext is: PTPTPT
Random key: (multiple) and it is used as the start point of the first
   chain.
After encrypting the plaintext T times (1048576 = 2^20, ) the chain end
   point becomes : (multiple)
Ciphertext Challenge: (multiple)
CT decrypts back to: (multiple)
output log: (multiple)
Output size (printf overhead): about 3170 lines, 323422 characters.
error log: (multiple)
```

```
Job log: (multiple)
source code file:(multiple)
job submission script:(multiple)
object file:(multiple)
Cores: 1024
usableCores: 1023
PID: (multiple) ....
Date: (multiple)
M= 1,072,693,248 = 2^30 records
C= 1024
W= 4 hours
T= 4096 = 2^12 iterations
B= 48
E= 2
H= 0
L= 0
S= 1
R= GPC
VLowD4096t1024c1072693248m8hrs_bestCvrg0
```

Observation: Job 46040593 found the Key but also found 7 false positive leads. 3 unique and two pairs with the same values. They all occurred at different t,c, and m.

```
VLowD4096t1024c1072693248m8hrs_bestCvrg0.o46040593
46040593

Lead at t=1461 core=207, m=1701912, key1stB_F_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[M-^?][ff]__[M-&][a6]__[6][36]__[^N][0e]__[M-z][fa]__[M-J][ca]_
Recovered Key(EP-1):_[F][46]__[M-x][f8]__[M-0][b0]__[^^][1e]__[a][61]__[U
   ][55]_
Chain SP:_[M-I][c9]__[^E][05]__[^X][18]__[M][4d]__[M-|][fc]__[M-^K][8b]_
Orig CT:_[^K][0b]__[M-8][b8]__[+][2b]__[^G][07]__[H][48]__[.][2e]_
Orig Key:_[3][33]__[5][35]__[g][67]__[t][74]__[w][77]__[F][46]_
CT 4 verify:_[M-@][c0]__[n][6e]__[,][2c]__[M-R][d2]__[M-_][df]__[^V][16]_

Lead at t=1160 core=791, m=683844, key1stB_^L_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
```

```
Found CT:_[j][6a]__[M-^I][89]__[R][52]__[-][2d]__[,][2c]__[M-W][d7]_
Recovered Key(EP-1):_[^L][0c]__[M-}][fd]__[L][4c]__[M-:][ba]__[M-J][ca]__[
   M-C][c3]_
Chain SP:_['][27]__[M-i][e9]__[M-#][a3]__[|][7c]__[^_][1f]__[.][2e]_
Orig CT:_[^K][0b]__[M-8][b8]__[+][2b]__[^G][07]__[H][48]__[.][2e]_
Orig Key:_[3][33]__[5][35]__[g][67]__[t][74]__[w][77]__[F][46]_
CT 4 verify:_[s][73]__[^G][07]__[M-$][a4]__[M-)][a9]__[M-^^][9e]__[z][7a]_

Lead at t=1151 core=515, m=636744, key1stB_=_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[$][24]__[=][3d]__[M-n][ee]__[;][3b]__[e][65]__[Z][5a]_
Recovered Key(EP-1):_[=][3d]__[M-^Z][9a]__[M-t][f4]__[,][2c]__[M-j][ea]__[
   J][4a]_
Chain SP:_[H][48]__[M-/][af]__[M-^C][83]__[m][6d]__[M-,][ac]__[^V][16]_
Orig CT:_[^K][0b]__[M-8][b8]__[+][2b]__[^G][07]__[H][48]__[.][2e]_
Orig Key:_[3][33]__[5][35]__[g][67]__[t][74]__[w][77]__[F][46]_
CT 4 verify:_[^L][0c]__[M-q][f1]__[^Y][19]__[`][60]__[M-U][d5]__[M-c][e3]_

Lead at t=931 core=40, m=211602, key1stB_;_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[M-]][dd]__[M-h][e8]__[M-\][dc]__[.][2e]__[M->][be]__[M-U][d5]_
Recovered Key(EP-1):_[;][3b]__[F][46]__[~][7e]__[M-a][e1]__[M-^?][ff]__[M-
   U][d5]_
Chain SP:_[^T][14]__[D][44]__[M-+][ab]__[`][60]__[M-n][ee]__[#][23]_
Orig CT:_[^K][0b]__[M-8][b8]__[+][2b]__[^G][07]__[H][48]__[.][2e]_
Orig Key:_[3][33]__[5][35]__[g][67]__[t][74]__[w][77]__[F][46]_
CT 4 verify:_[^S][13]__[,][2c]__[^F][06]__[,][2c]__[M-o][ef]__[H][48]_

Lead at t=275 core=706, m=894906, key1stB_=_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[y][79]__[M-%][a5]__[o][6f]__[L][4c]__[^\][1c]__[M-t][f4]_
Recovered Key(EP-1):_[=][3d]__[M-^Z][9a]__[M-t][f4]__[,][2c]__[M-j][ea]__[
   J][4a]_
Chain SP:_[M-u][f5]__[R][52]__[%][25]__[^T][14]__[^K][0b]__[M-C][c3]_
Orig CT:_[^K][0b]__[M-8][b8]__[+][2b]__[^G][07]__[H][48]__[.][2e]_
Orig Key:_[3][33]__[5][35]__[g][67]__[t][74]__[w][77]__[F][46]_
CT 4 verify:_[^L][0c]__[M-q][f1]__[^Y][19]__[`][60]__[M-U][d5]__[M-c][e3]_
```

```
Lead at t=452 core=456, m=1083252, key1stB_Q_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[M-|][fc]__[z][7a]__[M][4d]__[M-S][d3]__[^T][14]__[1][31]_
Recovered Key(EP-1):_[Q][51]__[M-0][b0]__[M-g][e7]__[M-^K][8b]__[M-Y][d9]
   __[M-^V][96]_
Chain SP:_[|][7c]__[M-@][c0]__[M-[][db]__[M-0][b0]__[M-^G][87]__[M-i][e9]_
Orig CT:_[^K][0b]__[M-8][b8]__[+][2b]__[^G][07]__[H][48]__[.][2e]_
Orig Key:_[3][33]__[5][35]__[g][67]__[t][74]__[w][77]__[F][46]_
CT 4 verify:_[~][7e]__[M-J][ca]__[^P][10]__[M-^[][9b]__[M-4][b4]__[#][23]_

Lead at t=226 core=1015, m=912840, key1stB_;_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[^T][14]__[^Y][19]__[O][4f]__[A][41]__[~][7e]__[M-B][c2]_
Recovered Key(EP-1):_[;][3b]__[F][46]__[~][7e]__[M-a][e1]__[M-^?][ff]__[M-
   U][d5]_
Chain SP:_[^D][04]__[M-+][ab]__[    ][09]__[|][7c]__[M-^X][98]__[}][7d]_
Orig CT:_[^K][0b]__[M-8][b8]__[+][2b]__[^G][07]__[H][48]__[.][2e]_
Orig Key:_[3][33]__[5][35]__[g][67]__[t][74]__[w][77]__[F][46]_
CT 4 verify:_[^S][13]__[,][2c]__[^F][06]__[,][2c]__[M-o][ef]__[H][48]_

Lead at t=1 core=1, m=0, key1stB_3_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[M-s][f3]__[^_][1f]__[M-u][f5]__[M-[][db]__[M-^]][9d]__[^K][0b]_
Recovered Key(EP-1):_[3][33]__[5][35]__[g][67]__[t][74]__[w][77]__[F][46]_
Chain SP:_[3][33]__[5][35]__[g][67]__[t][74]__[w][77]__[F][46]_
Orig CT:_[^K][0b]__[M-8][b8]__[+][2b]__[^G][07]__[H][48]__[.][2e]_
Orig Key:_[3][33]__[5][35]__[g][67]__[t][74]__[w][77]__[F][46]_
CT 4 verify:_[^K][0b]__[M-8][b8]__[+][2b]__[^G][07]__[H][48]__[.][2e]_
We have Verification also.

Limits: nednodes=128:ppn=8,nodes=128:ppn=8,walltime=08:00:00
Resources:     cput=1211:39:19,mem=209164436kb,vmem=394810176kb,walltime
   =01:11:30
```

Job ID 46040577 46040579 46040581 46040594 46040595 46040596 46040597

```
The number of booked cores is 1024 (1 Parent and 1023 children)
```

```
plaintext is: PTPTPT
Random key: (multiple) and it is used as the start point of the first
   chain.
After encrypting the plaintext T times (1048576 = 2^20, ) the chain end
   point becomes : (multiple)
Ciphertext Challenge: (multiple)
CT decrypts back to: (multiple)
output log: (multiple)
Output size (printf overhead): about 3170 lines, 323422 characters.
error log: (multiple)
Job log: (multiple)
source code file:(multiple)
job submission script:(multiple)
object file:(multiple)
Cores: 1024
usableCores: 1023
PID: (multiple) ....
Date: (multiple)
M= 2,145,386,496 = 2^31 records
C= 1024
W= 4 hours
T= 4096 = 2^12 iterations
B= 48
E= 2
H= 0
L= 0
S= 1
R= GPC
VLowD4096t1024c1072693248m8hrs_bestCvrg0
```

Observation: These set of jobs finished without finding the key, exhausting the walltime or issuing an error. Memory limits are assumed to be good since initializing the SP was fine. Looking at the allocated memory, it is about 227gb if we take the median 238314532kb.

```
VLowD 4096t 1024c 2145386496m 4hrs_bestCvrg0.o46040577
46040577
Limits: nednodes=128:ppn=8,nodes=128:ppn=8,walltime=04:00:00
Resources:    cput=2373:31:14,mem=238526400kb,vmem=424213184kb,walltime
```

```
   =02:19:57

VLowD 4096t 1024c 2145386496m 4hrs_bestCvrg0.o46040579
46040579
Limits: neednodes=128:ppn=8,nodes=128:ppn=8,walltime=04:00:00
Resources:     cput=2373:24:27,mem=238154188kb,vmem=424014920kb,walltime
   =02:19:49

VLowD 4096t 1024c 2145386496m 4hrs_bestCvrg0.o46040581
46040581
Limits: neednodes=128:ppn=8,nodes=128:ppn=8,walltime=04:00:00
Resources:     cput=2373:18:04,mem=238314532kb,vmem=424103584kb,walltime
   =02:19:52

VLowD 4096t 1024c 2145386496m 8hrs_bestCvrg0.o46040594
46040594
Limits: neednodes=128:ppn=8,nodes=128:ppn=8,walltime=08:00:00
Resources:     cput=2373:22:05,mem=238535460kb,vmem=424220460kb,walltime
   =02:19:52

VLowD 4096t1024c 2145386496m 8hrs_bestCvrg0.o46040595
46040595
Limits: neednodes=128:ppn=8,nodes=128:ppn=8,walltime=08:00:00
Resources:     cput=2373:24:22,mem=238136416kb,vmem=424006248kb,walltime
   =02:19:59

VLowD 4096t 1024c 2145386496m 8hrs_bestCvrg0.o46040596
46040596
Limits: neednodes=128:ppn=8,nodes=128:ppn=8,walltime=08:00:00
Resources:     cput=2373:31:33,mem=238529352kb,vmem=424218176kb,walltime
   =02:19:56

VLowD 4096t 1024c 2145386496m 8hrs_bestCvrg0.o46040597
46040597
Limits: neednodes=128:ppn=8,nodes=128:ppn=8,walltime=08:00:00
Resources:     cput=2373:18:24,mem=238083204kb,vmem=423983908kb,walltime
   =02:19:53
```

A.2.4 Changing the algorithm

On of the objectives of the research was to compare similar runs in terms of key variables using different but comparable encryption algorithms. Since Simeck is very similar to Speck it was the obvious choice. Job ID: 46336986 (46336988 46336987 46336985 did not succeed)

```
The number of booked cores is 1024 (1 Parent and 1023 children)
plaintext is: PTPTPT
Random key: (multiple) and it is used as the start point of the first
   chain.
After encrypting the plaintext T times (1048576 = 2^20, ) the chain end
   point becomes : (multiple)
Ciphertext Challenge: (multiple)
CT decrypts back to: (multiple)
output log: (multiple)
Output size (printf overhead): about 22045 lines, 5390223 characters.
error log: (multiple)
Job log: (multiple)
source code file:(multiple)
job submission script:(multiple)
object file:(multiple)
Cores: 1024
usableCores: 1023
PID: (multiple) ....
Date: (multiple)
M= 1,072,693,248 = 2^30 records
C= 1024
W= 4 hours
T= 4096 = 2^12 iterations
B= 48
E= 3 Speck
H= 0
L= 0
S= 1
R= GPC
Notes: SpeckVLowD4096t1024c2145386496m4hrs_bestCvrg0
```

```
lead at t=1 core=1, m=0, key1stB_0_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[,][2c]__[^][5e]__[M-o][ef]__[7][37]__[M-l][ec]__[^@][00]_
Recovered Key(EP-1):_[O][4f]__[P][50]__[x][78]__[b][62]__[x][78]__[h][68]_
Chain SP:_[O][4f]__[P][50]__[x][78]__[b][62]__[x][78]__[h][68]_
Orig CT:_[M-5][b5]__[M-n][ee]__[M-R][d2]__[7][37]__[J][4a]__[^@][00]_
Orig Key:_[O][4f]__[P][50]__[x][78]__[b][62]__[x][78]__[h][68]_
CT 4 verify:_[M-5][b5]__[M-n][ee]__[M-R][d2]__[7][37]__[J][4a]__[^@][00]_
We have Verification also.

    ctime = Thu Dec 21 09:46:56 2017
    mtime = Thu Dec 21 09:46:56 2017
    qtime = Thu Dec 21 09:46:56 2017
    Resource_List.walltime = 04:00:00
    etime = Thu Dec 21 09:46:56 2017
```

Observation: Four tests were run using Speck and only one of them succeeded in retrieving the verification key. The rate of false positives was very high comparing to Simeck which lead to higher output density and the inability to compare run times.

A.2.5 Measuring the effect of sort

The complexity of sorting algorithms have a significant penalty on the runs that are not particularly useful in this phase of the research as it entails comparing many runs. However, it was also important to sample some runs with sorting enabled to draw a comparison. The following jobs are comparable to jobs that had high output density for Simeck. The general observation from those jobs that work that used to take a less than two hours , now needs more than 8 hours. Indeed the program gets kicked out by GPC while sorting.

```
45940946
=>> PBS: job killed: walltime 14408 exceeded limit 14400
Limits: nednodes=128:ppn=8,nodes=128:ppn=8,walltime=04:00:00
Resources:     cput=4093:09:25,mem=206990532kb,vmem=393684032kb,walltime
   =04:00:08

45940947
=>> PBS: job killed: walltime 14413 exceeded limit 14400
Limits: nednodes=128:ppn=8,nodes=128:ppn=8,walltime=04:00:00
```

```
Resources:     cput=4096:06:50,mem=206990448kb,vmem=393684004kb,walltime
   =04:00:13

45940948
=>> PBS: job killed: walltime 14422 exceeded limit 14400
Limits: neednodes=128:ppn=8,nodes=128:ppn=8,walltime=04:00:00
Resources:     cput=4097:31:34,mem=206983836kb,vmem=393684028kb,walltime
   =04:00:22

45940949
=>> PBS: job killed: walltime 14436 exceeded limit 14400
Limits: neednodes=128:ppn=8,nodes=128:ppn=8,walltime=04:00:00
Resources:     cput=4101:34:31,mem=206983616kb,vmem=393684004kb,walltime
   =04:00:36

45940950
=>> PBS: job killed: walltime 14430 exceeded limit 14400
Limits: neednodes=128:ppn=8,nodes=128:ppn=8,walltime=04:00:00
Resources:     cput=4101:32:38,mem=206983116kb,vmem=393684000kb,walltime
   =04:00:30

45940951
=>> PBS: job killed: walltime 14416 exceeded limit 14400
Limits: neednodes=128:ppn=8,nodes=128:ppn=8,walltime=04:00:00
Resources:     cput=4096:38:56,mem=206970196kb,vmem=393683924kb,walltime
   =04:00:16

45940952
=>> PBS: job killed: walltime 14407 exceeded limit 14400
Limits: neednodes=128:ppn=8,nodes=128:ppn=8,walltime=04:00:00
Resources:     cput=4094:17:48,mem=206970208kb,vmem=393684080kb,walltime
   =04:00:07

45940953
=>> PBS: job killed: walltime 14427 exceeded limit 14400
Limits: neednodes=128:ppn=8,nodes=128:ppn=8,walltime=04:00:00
Resources:     cput=4100:37:55,mem=206973392kb,vmem=393683840kb,walltime
   =04:00:27
```

```
45940954
=>> PBS: job killed: walltime 14427 exceeded limit 14400
Limits: neednodes=128:ppn=8,nodes=128:ppn=8,walltime=04:00:00
Resources:     cput=4098:47:13,mem=206984756kb,vmem=393683928kb,walltime
   =04:00:28

45940955
=>> PBS: job killed: walltime 14416 exceeded limit 14400
Limits: neednodes=128:ppn=8,nodes=128:ppn=8,walltime=04:00:00
Resources:     cput=4095:19:13,mem=206990148kb,vmem=393684076kb,walltime
   =04:00:17

45940966
=>> PBS: job killed: walltime 28827 exceeded limit 28800
Limits: neednodes=128:ppn=8,nodes=128:ppn=8,walltime=08:00:00
Resources:     cput=8199:35:42,mem=206989880kb,vmem=393683956kb,walltime
   =08:00:27

45940967
=>> PBS: job killed: walltime 28815 exceeded limit 28800
Limits: neednodes=128:ppn=8,nodes=128:ppn=8,walltime=08:00:00
Resources:     cput=8196:21:27,mem=206991164kb,vmem=393684048kb,walltime
   =08:00:16

45940968
=>> PBS: job killed: walltime 28841 exceeded limit 28800
Limits: neednodes=128:ppn=8,nodes=128:ppn=8,walltime=08:00:00
Resources:     cput=8203:20:29,mem=206988500kb,vmem=393683904kb,walltime
   =08:00:41

45940969
=>> PBS: job killed: walltime 28830 exceeded limit 28800
Limits: neednodes=128:ppn=8,nodes=128:ppn=8,walltime=08:00:00
Resources:     cput=8198:48:40,mem=206988120kb,vmem=393683992kb,walltime
   =08:00:30

45940970
```

```
=>> PBS: job killed: walltime 28836 exceeded limit 28800
Limits: neednodes=128:ppn=8,nodes=128:ppn=8,walltime=08:00:00
Resources:     cput=8199:08:41,mem=206990172kb,vmem=393683920kb,walltime
   =08:00:36

45940971
=>> PBS: job killed: walltime 28812 exceeded limit 28800
Limits: neednodes=128:ppn=8,nodes=128:ppn=8,walltime=08:00:00
Resources:     cput=8195:11:47,mem=206989844kb,vmem=393683848kb,walltime
   =08:00:12

45940972
=>> PBS: job killed: walltime 28815 exceeded limit 28800
Limits: neednodes=128:ppn=8,nodes=128:ppn=8,walltime=08:00:00
Resources:     cput=8195:31:35,mem=206989220kb,vmem=393683968kb,walltime
   =08:00:15

45940973
=>> PBS: job killed: walltime 28810 exceeded limit 28800
Limits: neednodes=128:ppn=8,nodes=128:ppn=8,walltime=08:00:00
Resources:     cput=8194:41:48,mem=206989324kb,vmem=393683996kb,walltime
   =08:00:10

45940974
=>> PBS: job killed: walltime 28820 exceeded limit 28800
Limits: neednodes=128:ppn=8,nodes=128:ppn=8,walltime=08:00:00
Resources:     cput=8196:48:44,mem=206986328kb,vmem=393684012kb,walltime
   =08:00:20

45940975
=>> PBS: job killed: walltime 28817 exceeded limit 28800
Limits: neednodes=128:ppn=8,nodes=128:ppn=8,walltime=08:00:00
Resources:     cput=8195:47:16,mem=206986944kb,vmem=393684076kb,walltime
   =08:00:17

45940956
=>> PBS: job killed: walltime 14422 exceeded limit 14400
Limits: neednodes=128:ppn=8,nodes=128:ppn=8,walltime=04:00:00
```

```
Resources:     cput=4096:22:18,mem=236318900kb,vmem=423044296kb,walltime
   =04:00:22

45940957
=>> PBS: job killed: walltime 14421 exceeded limit 14400
Limits: neednodes=128:ppn=8,nodes=128:ppn=8,walltime=04:00:00
Resources:     cput=4097:03:52,mem=236318968kb,vmem=423044144kb,walltime
   =04:00:21

45940958
=>> PBS: job killed: walltime 14408 exceeded limit 14400
Limits: neednodes=128:ppn=8,nodes=128:ppn=8,walltime=04:00:00
Resources:     cput=4094:20:12,mem=236323900kb,vmem=423044004kb,walltime
   =04:00:08

45940959
=>> PBS: job killed: walltime 14424 exceeded limit 14400
Limits: neednodes=128:ppn=8,nodes=128:ppn=8,walltime=04:00:00
Resources:     cput=4098:08:55,mem=236322412kb,vmem=423044208kb,walltime
   =04:00:24

45940960
=>> PBS: job killed: walltime 14411 exceeded limit 14400
Limits: neednodes=128:ppn=8,nodes=128:ppn=8,walltime=04:00:00
Resources:     cput=4095:13:21,mem=236318832kb,vmem=423044088kb,walltime
   =04:00:11

45940961
=>> PBS: job killed: walltime 14435 exceeded limit 14400
Limits: neednodes=128:ppn=8,nodes=128:ppn=8,walltime=04:00:00
Resources:     cput=4101:54:49,mem=236331248kb,vmem=423044016kb,walltime
   =04:00:35

45940962
=>> PBS: job killed: walltime 14434 exceeded limit 14400
Limits: neednodes=128:ppn=8,nodes=128:ppn=8,walltime=04:00:00
Resources:     cput=4100:29:44,mem=236320120kb,vmem=423043988kb,walltime
   =04:00:34
```

```
45940963
=>> PBS: job killed: walltime 14445 exceeded limit 14400
Limits: neednodes=128:ppn=8,nodes=128:ppn=8,walltime=04:00:00
Resources:     cput=4103:44:47,mem=236318120kb,vmem=423044144kb,walltime
   =04:00:45

45940964
=>> PBS: job killed: walltime 14412 exceeded limit 14400
Limits: neednodes=128:ppn=8,nodes=128:ppn=8,walltime=04:00:00
Resources:     cput=4095:05:21,mem=236319680kb,vmem=423044144kb,walltime
   =04:00:12

45940965
=>> PBS: job killed: walltime 14443 exceeded limit 14400
Limits: neednodes=128:ppn=8,nodes=128:ppn=8,walltime=04:00:00
Resources:     cput=4103:45:21,mem=236337376kb,vmem=423043868kb,walltime
   =04:00:43

45940976
=>> PBS: job killed: walltime 28825 exceeded limit 28800
Limits: neednodes=128:ppn=8,nodes=128:ppn=8,walltime=08:00:00
Resources:     cput=8197:49:38,mem=236337748kb,vmem=423044120kb,walltime
   =08:00:26

45940977
=>> PBS: job killed: walltime 28838 exceeded limit 28800
Limits: neednodes=128:ppn=8,nodes=128:ppn=8,walltime=08:00:00
Resources:     cput=8200:59:39,mem=236330388kb,vmem=423044088kb,walltime
   =08:00:38

45940978
=>> PBS: job killed: walltime 28804 exceeded limit 28800
Limits: neednodes=128:ppn=8,nodes=128:ppn=8,walltime=08:00:00
Resources:     cput=8191:37:20,mem=236335732kb,vmem=423044128kb,walltime
   =08:00:05

45940979
```

```
=>> PBS: job killed: walltime 28819 exceeded limit 28800
Limits: neednodes=128:ppn=8,nodes=128:ppn=8,walltime=08:00:00
Resources:     cput=8195:09:43,mem=236319760kb,vmem=423044204kb,walltime
   =08:00:19

45940980
=>> PBS: job killed: walltime 28807 exceeded limit 28800
Limits: neednodes=128:ppn=8,nodes=128:ppn=8,walltime=08:00:00
Resources:     cput=8192:42:38,mem=236323992kb,vmem=423044056kb,walltime
   =08:00:08

45940981
=>> PBS: job killed: walltime 28830 exceeded limit 28800
Limits: neednodes=128:ppn=8,nodes=128:ppn=8,walltime=08:00:00
Resources:     cput=8199:50:42,mem=236337904kb,vmem=423044104kb,walltime
   =08:00:30

45940982
=>> PBS: job killed: walltime 28840 exceeded limit 28800
Limits: neednodes=128:ppn=8,nodes=128:ppn=8,walltime=08:00:00
Resources:     cput=8202:23:16,mem=236337920kb,vmem=423044080kb,walltime
   =08:00:40

45940983
=>> PBS: job killed: walltime 28834 exceeded limit 28800
Limits: neednodes=128:ppn=8,nodes=128:ppn=8,walltime=08:00:00
Resources:     cput=8199:57:13,mem=236337616kb,vmem=423043908kb,walltime
   =08:00:34

45940984
=>> PBS: job killed: walltime 28821 exceeded limit 28800
Limits: neednodes=128:ppn=8,nodes=128:ppn=8,walltime=08:00:00
Resources:     cput=8195:41:41,mem=236338448kb,vmem=423043924kb,walltime
   =08:00:21

45940985
=>> PBS: job killed: walltime 28810 exceeded limit 28800
Limits: neednodes=128:ppn=8,nodes=128:ppn=8,walltime=08:00:00
```

```
Resources:     cput=8192:57:07,mem=236323480kb,vmem=423044028kb,walltime
   =08:00:11
```

```
16sort1024t1024c1072693248m16hrs_bestCvrg0.o45996933
================================================================
45996933
Lead at t=275 core=939, m=2072682, key1stB_M-}_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[^@][00]__[o][6f]__[8][38]__[A][41]__[M--][ad]__[‘][60]_
Recovered Key(EP-1):_[M-}][fd]__[^G][07]__[Q][51]__[8][38]__[;][3b]__[M
   -6][b6]_
Chain SP:_[L][4c]__[M-^G][87]__[M-w][f7]__[s][73]__[c][63]__[M-^D][84]_
Orig CT:_[M-^Y][99]__[M->][be]__[%][25]__[M-^O][8f]__[M-1][b1]__[f][66]_
Orig Key:_[y][79]__[D][44]__[r][72]__[h][68]__[L][4c]__[X][58]_
CT 4 verify:_[_][5f]__[M-)][a9]__[M-e][e5]__[\][5c]__[r][72]__[m][6d]_
Limits: nodes=128:ppn=8,nodes=128:ppn=8,walltime=16:00:00
Resources:     cput=12908:53:15,mem=207530608kb,vmem=393683776kb,walltime
   =12:40:12

16sort1024t1024c1072693248m16hrs_bestCvrg0.o45996934
================================================================
45996934
Lead at t=392 core=524, m=41292, key1stB_M-^E_
Orig PT:_[P][50]__[T][54]__[P][50]__[T][54]__[P][50]__[T][54]_
Found CT:_[^@][00]__[M-m][ed]__[M-^][de]__[(][28]__[L][4c]__[M-‘][e0]_
Recovered Key(EP-1):_[M-^E][85]__[^F][06]__[^T][14]__[X][58]__[^D][04]__[M
   -.][ae]_
Chain SP:_[M-^J][8a]__[k][6b]__[M-^F][86]__[M-A][c1]__[M-[][db]__[’][27]_
Orig CT:_[M-J][ca]__[M-8][b8]__[]][5d]__[^@][00]__[M-z][fa]__[^@][00]_
Orig Key:_[e][65]__[B][42]__[S][53]__[J][4a]__[w][77]__[k][6b]_
CT 4 verify:_[Z][5a]__[M-\][dc]__[^\][1c]__[T][54]__[h][68]__[M-H][c8]_
Limits: nodes=128:ppn=8,nodes=128:ppn=8,walltime=16:00:00
Resources:     cput=12912:20:45,mem=207527412kb,vmem=393683796kb,walltime
   =12:41:13
```

Appendix B

Shell Code

B.1 Shell code used to submit the compiled C code (object code)

```
tmt.sh
#!/bin/bash
# MOAB/Torque submission script for SciNet GPC
#
#PBS -l nodes=256:ppn=8,walltime=48:00:00
#PBS -N pushingCores
#PBS -m abe

# load modules (must match modules used for compilation)
module load intel/15.0.2 openmpi/intel/1.6.4

# DIRECTORY TO RUN - $PBS_O_WORKDIR is directory job was submitted from
cd $PBS_O_WORKDIR

# EXECUTION COMMAND; -np = nodes*ppn
mpirun -np 2048 ./tmt.out
```

B.2 Shell code used to run a range of permutations (multiple sizes and main variables)

```
48_tmt_test.sh

# Pushing M
# records per core = 2 GB / (2 records * 32BS/8 ) - overhead = 2 1024 1024
    1024 / 8 = 512 M record pre core = 500 million recorde per core. 300
   million if 48bit
# we are now at 1 million. we need to test 1 to 500 million records per
   core
# eventually for BS48 we can atim for 2^28 = 268435456 = less than 300
   million record = 2,147,483,648 bytes
# Pushing C
# Without special permission i'm allowed to submit #PBS -l nodes=256:ppn
   =8,walltime=48:00:00 which means maximum of 2048 cores
# we should test 15 30 45 60 2 3 4 ... 48 . righ now we can keep it at 30
   minute

# Pushing T though W (walltime)
# without special permission i'm allowed to submit request for 48 hours
# now, 15 minutes is barely enough for t=1024 and m=1024*1024. 15*60=
   900=~ 1024. We can assume seconds of walltime = t*m / 1024

# 48 hr* 60min * 60 sec = 172800 (2^17 131072).... 900sec make 1G m*t ~~~
   1024 second 1024 1024 1024 tm

#w=30;
#c=1024;
s=1;
let "z=2**48"
for w in 4 ; do # maximum time in batchmode is 48... 04 08 16 24 32 40 48;
    do # should be in hours `seq 1 48`       # 2^10?
  for c in 1024;do #1024 2048; do # 2048    # 2^11          Cores stop
     working on 1024, 832 ran out of walltime but essentially fine
    for t in 4096 1048576; do # it seems 2^20 1048576 works ###48bit 204857
        worked, but 304857 didnt find the winner... # can we get to 2^18
```

```
       204857
     for f in 1 2 ;do # 2 4; do           # 2^8   #128 and above runs out
        of memory, 8 gets An error occurred in MPI_Type_vector, 2 is fine
        , 4 exceeds 1hour
       m=‘expr $(expr $f) \* 1024 \* 1024 \* $( expr $c - 1 )‘ ; # 2^20
       mt=‘expr $m \* $t / $z‘;
       echo "Walltime=$w , Cores+1=$c , T=$t , mFactor= $f , Total M for
          all Cores= $m .... Key space= $z , Best case Coverage $mt....
          NoSort=${s}" ;
         for test in ‘seq 1 10‘; do
         KEY=$(cat /dev/urandom | tr -dc ’a-zA-Z0-9’ | fold -w 6 | head -n
             1 ) ;
##KEY="KCKCKC";
         KEYHEX=$(for i in ‘echo $KEY | sed -e ’s/\(.\)/\1\n/g’ ‘ ; do
            printf ’%02x’ "’$i" ; done | grep "");
         PTEXT=$(cat /dev/urandom | tr -dc ’a-zA-Z0-9’ | fold -w 6 | head
            -n 1 ) ;
         #echo "KEY   $KEY";
         #echo "HEX   $KEYHEX";
         #echo "Ptext $PTEXT";

    ./tmt_create.sh -M ${m} -C ${c} -W ${w} -T ${t} -B 48 -E 2 -H 0 -L 0 -
       S 1 -R GPC -K $KEY -X $KEYHEX -P PTPTPT -n IncT${t}t${c}c${m}m${w}
       hrs_bestCvrg${mt} ;
echo ./tmt_create.sh -M ${m} -C ${c} -W ${w} -T ${t} -B 48 -E 2 -H 0 -L 0
   -S 1 -R GPC -K $KEY -X $KEYHEX -P PTPTPT -n tmto${t}t${c}c${m}m${w}
   hrs_bestCvrg${mt} ;
       done
     done
   done
  done
done
```

B.3 Shell code used to collect data from runs and saving them in csv format to be plotted in spread sheets

```
tmt_analyze.sh
#
#ls *Push_*.c;v
TAG="Oct_"
echo "Comment; M; C; W; T; B; E; H; L; S; R; N; Winners; resources; limits
   ; error; leads; verifications; MAXCOV; Clog2; Mlog2; Tlog2; resources;
   cput; mem; vmem; walltime1; limits; nednodes; ppn1; nodes; ppn2;
   walltime2; outputDensityLines; outputDensityChars; o_file; e_file;
   cfile; ofile; sfile; lfile; wfile; GREPME; Verified;" >
   tmt_results_table_${TAG}.csv
echo $SCRATCH
let "fn=0";
for cfile in `ls -t ./tmt_*${TAG}*.c| head -900 `; do
  fn=`expr $fn + 1`; echo " "; echo " "; echo " "; echo " ";
  echo "file#: $fn ============================================="
  echo "cfile: $cfile"
  echo "..................................................."
  ofile=${cfile/.c/.out}; #http://stackoverflow.com/questions/13210880/
     replace-one-substring-for-another-string-in-shell-script
  if [ -a ${HOME}/${ofile} ]
  then
    printf "1>>>> object file exists:\n ${HOME}/${ofile} \n";
  else
    echo "object file DOESN'T exists: ${HOME}/${ofile} ";
  fi

  echo "..................................................."
  sfile=${cfile/.c/.sh};
  if [ -a ${HOME}/${sfile} ]
  then
    printf "2>>>> shell file exists:\n ${HOME}/${sfile} \n";
  else
```

```
    echo "shell file DOESN'T exists: ${HOME}/${sfile} ";
  fi
  echo "....................................................."

  lfile=${cfile/.c/\*.joblog};
  #ls -l ${lfile}
  if [ -a ${HOME}/${lfile} ]
  then
    printf "3>>>> job log file exists:\n ${HOME}/${lfile} \n";
  else
    echo "job log DOESN'T exists: ${HOME}/${lfile} ";
  fi
  echo "....................................................."

  wfile=${cfile/.c/.winner.txt};
  #echo "wfile $wfile"
  if [ -a ${SCRATCH}/${wfile} ]
  then
    printf "4>>>> Winner file exists:\n ${SCRATCH}/${wfile} \n";
    winners=`wc -l < ${SCRATCH}/${wfile}`; #http://stackoverflow.com/
       questions/10238363/how-to-get-wc-l-to-print-just-the-number-of-
       lines-without-file-name
    cat ${SCRATCH}/${wfile}
  else
    echo "winner file DOESN'T exists: $wfile ";
    winners="0";
  fi
  echo "....................................................."

# ./tmt_Sat_Jul__1_05_51_43_EDT_2017_M=535822336_C=512_W=01_00_00_T=1024_B
    =48_E=2_H=0_L=0_S=1_R=GPC__notes=
    Z48Z_ns1024t512c535822336m01hrs_bestCvrg0_43108853.gpc-sched-ib0.joblog
#tmt_Mon_Apr_10_07_51_17_EDT_2017_M=535822336_C=512_W=00_30_00_T=64_B=32_E
    =2_H=0_L=0_S=1_R=GPC__notes=Push_ns64t512c535822336m30min
  notes=`echo $cfile | gawk -F'__notes=' '{ print $2; } ' | sed -e 's/\.c$
     //' `
  jobid=`echo $lfile | gawk -F'_bestCvrg' '{ print $2; } ' | sed -e 's/\.
     gpc.*$//' | sed -e 's/.*_//'`
```

```
  printf ">>>> Notes are: $notes \n
     ..................................................... \n JObID is:
     $jobid \n..................................................... \n";
#  ls ${notes}.o${jobid} ;
#  ls $SCRATCH/${notes}.o${jobid} ;

#test -f $SCRATCH/${notes}.o${jobid} ; echo "==>> $SCRATCH/${notes}.o${
   jobid} ===>>> $?";

  o_file=$HOME/${notes}.o${jobid}
  o_file2=$HOME/${notes}.o${jobid}
  #echo $o_file;
  if [ -e ${o_file} ]
  then
    printf "5>>>> output file exists:\n ${o_file} \n";

    resources=`cat -v ${o_file}| grep Resources`
    #https://stackoverflow.com/questions/10520623/how-to-split-one-string-
       into-multiple-variables-in-bash-shell
    cput=$(echo $resources | cut -f1 -d, |cut -f2 -d=)
    mem=$(echo $resources | cut -f2 -d, | cut -f2 -d= | cut -f1 -dk)
    vmem=$(echo $resources | cut -f3 -d, | cut -f2 -d= | cut -f1 -dk)
    walltime1=$(echo $resources | cut -f4 -d, | cut -f2 -d=)
echo $resources
echo $cput
echo $mem
echo $vmem
echo $walltime1


    limits=`cat -v ${o_file}| grep Limits `
    neednodes=$(echo $limits | cut -f1 -d, |cut -f2 -d: |cut -f2 -d=)
    ppn1=$(echo $limits | cut -f1 -d, |cut -f3 -d: |cut -f2 -d=)
    nodes=$(echo $limits | cut -f2 -d, |cut -f1 -d: |cut -f2 -d=)
    ppn2=$(echo $limits | cut -f2 -d, |cut -f2 -d: |cut -f2 -d=)
    walltime2=$(echo $limits | cut -f3 -d, |cut -f2 -d=)
echo $limits
echo $neednodes
```

```
echo $ppn1
echo $nodes
echo $ppn2
echo $walltime2

outputDensityLines=‘wc -l < ${o_file}‘
outputDensityChars=‘wc -c < ${o_file}‘

   leads=‘cat -v ${o_file}| grep Yaaaay | wc -l‘
   verified=‘cat -v ${o_file}| grep Verification | head -1‘
   verifications=‘cat -v ${o_file}| grep Verification | wc -l‘
   grepper=‘cat -v ${o_file}| grep GREPME ‘

   #resources=‘grep Resources ${o_file}‘
   #limits=‘grep Limits ${o_file}‘
   #leads=‘grep Yaaaay ${o_file} | wc -l‘
   #verified=‘grep Verification ${o_file}| head -1‘
   #verifications=‘grep Verification ${o_file}| wc -l‘
   #grepper=‘grep GREPME ${o_file}‘
  elif [ -e ${o_file2} ]
  then
   #o_file=‘echo ${o_file2}‘
   printf "5,>>>> output file exists:\n ${o_file} \n";

   resources=‘cat -v ${o_file}| grep Resources‘
   cput=$(echo $resources | cut -f1 -d, |cut -f2 -d=)
   mem=$(echo $resources | cut -f2 -d, | cut -f2 -d= | cut -f1 -dk)
   vmem=$(echo $resources | cut -f3 -d, | cut -f2 -d= | cut -f1 -dk)
   walltime1=$(echo $resources | cut -f4 -d, | cut -f2 -d=)
echo $resources
echo $cput
echo $mem
echo $vmem
echo $walltime1



   limits=‘cat -v ${o_file}| grep Limits ‘
```

```
    neednodes=$(echo $limits | cut -f1 -d, |cut -f2 -d: |cut -f2 -d=)
    ppn1=$(echo $limits | cut -f1 -d, |cut -f3 -d: |cut -f2 -d=)
    nodes=$(echo $limits | cut -f2 -d, |cut -f1 -d: |cut -f2 -d=)
    ppn2=$(echo $limits | cut -f2 -d, |cut -f2 -d: |cut -f2 -d=)
    walltime2=$(echo $limits | cut -f3 -d, |cut -f2 -d=)

echo $limits
echo $neednodes
echo $ppn1
echo $nodes
echo $ppn2
echo $walltime2

outputDensityLines=‘wc -l < ${o_file}‘
outputDensityChars=‘wc -c < ${o_file}‘


    leads=‘cat -v ${o_file}| grep Yaaaay | wc -l‘
    verified=‘cat -v ${o_file}| grep Verification | head -1‘
    verifications=‘cat -v ${o_file}| grep Verification | wc -l‘
    grepper=‘cat -v ${o_file}| grep GREPME ‘


#    resources=‘grep Resources ${o_file}‘
#    limits=‘grep Limits ${o_file}‘
#    leads=‘grep Yaaaay ${o_file} | wc -l‘
#    verified=‘grep Verification ${o_file}| head -1‘
#    verifications=‘grep Verification ${o_file}| wc -l‘
#    grepper=‘grep GREPME ${o_file}‘
  else
    echo  "output file DOESN’t exists: ${o_file} ";
    resources="unclear"
    limits="unclear"
  fi
  echo "....................................................."

  e_file=${HOME}/${notes}.e${jobid}
  #echo $e_file;
```

```
if [ -e ${e_file} ]
then
  echo "6>>>> error file exists: ${e_file} ";
  wc -l ${e_file};
  error=‘grep ERR_ ${e_file} | sed -e ’s/.*\*//’‘
  #####if error=="" then error="unclear" .. wrong grep
else
  echo "error file DOESN’t exists: ${e_file} ";
  error="NoError"
fi
echo "................................................."




let "count=0";
for i in $(echo $cfile | tr "_" "\n" | tr "=" "\n")
do
  count=‘expr $count + 1‘;
  #echo -n $count $i;
      if [ $count -eq ’11’ ]; then
             M=$i;
             echo " M=’$M’";
      fi
      if [ $count -eq ’13’ ]; then
             C=$i;
             echo " C=’$C’";
             Cl2=‘echo "l($C)/l(2)" | bc -l‘ ;
             Clog2=‘echo ${Cl2%.*}‘
             MperC=‘expr $(expr $M) / $(expr $C - 1 ) ‘;
             Ml2=‘echo "l($MperC)/l(2)" | bc -l‘ ;
             Mlog2=‘echo ${Ml2%.*}‘

      fi
      if [ $count -eq ’15’ ]; then
             W=${i};
             echo " W=’$W’ ";
      fi
```

```
if [ $count -eq '16' ]; then
       W=${W}:${i};
       echo " W='$W' ";
fi
if [ $count -eq '17' ]; then
       W=${W}:${i};
       echo " W='$W'";
fi

if [ $count -eq '19' ]; then
       T=$i;
       echo " T='$T'";
       Tl2=`echo "l($T)/l(2)" | bc -l` ;
       Tlog2=`echo ${Tl2%.*}`
       MAXCOV=$(( $Clog2 + Tlog2 + Mlog2))
fi
if [ $count -eq '21' ]; then
       B=$i;
       echo " B='$B'";
fi
if [ $count -eq '23' ]; then
       E=$i;
       echo " E='$E'";
fi
if [ $count -eq '25' ]; then
       H=$i;
       echo " H='$H'";
fi
if [ $count -eq '27' ]; then
       L=$i;
       echo " L='$L'";
fi
if [ $count -eq '29' ]; then
       S=$i;
       echo " S='$S'";
fi
if [ $count -eq '31' ]; then
       R=$i;
```

```
                echo  " R='$R'";
        fi
        if [ $count -eq '33' ]; then
                N=$i;
                echo  " N='$N'";
        fi

  # process
  done
  echo " Win=${winners}"
  echo " Err=${error} "
  echo " Res=${resources}"
  echo " Lim=${limits}"
  echo " Leads= ${leads}"
  echo " Verifications=${verifications}"
  echo " verified=${verified}"
  echo " MaxCoverage=${MAXCOV}"
  echo " Clog2=${Clog2}"
  echo " Mlog2=${Mlog2}"
  echo " Tlog2=${Tlog2}"
  echo " grepper=${grepper}"
echo "........................................."


#  echo "${notes}; ${M}; ${C}; ${W}; ${T}; ${B}; ${E}; ${H}; ${L}; ${S}; $
   {R}; ${N}; ${winners}; ${resources}; ${limits}; ${error}; ${leads}; ${
   verifications}; ${verified}; ${o_file}; ${e_file}; ${cfile}; ${ofile};
   ${sfile}; ${lfile}; ${wfile};" >> tmt_results_table.csv
##  echo "${notes}; ${M}; ${C}; ${W}; ${T}; ${B}; ${E}; ${H}; ${L}; ${S}; $
   {R}; ${N}; ${winners}; ${resources}; ${limits}; ${error}; ${leads}; ${
   verifications}; ${verified}; ${MAXCOV}; ${Clog2}; ${Mlog2}; ${Tlog2}; $
   {grepper}"
 echo "${notes}; ${M}; ${C}; ${W}; ${T}; ${B}; ${E}; ${H}; ${L}; ${S}; ${R
    }; ${N}; ${winners}; ${resources}; ${limits}; ${error}; ${leads}; ${
    verifications}; ${MAXCOV}; ${Clog2}; ${Mlog2}; ${Tlog2}; $resources;
    $cput; $mem; $vmem; $walltime1; $limits; $neednodes; $ppn1; $nodes;
    $ppn2; $walltime2; $outputDensityLines; $outputDensityChars; ${o_file
    }; ${e_file}; ${cfile}; ${ofile}; ${sfile}; ${lfile}; ${wfile}; ${
```

```
    grepper}; ${verified}" >> tmt_results_table_${TAG}.csv
  echo "${notes}; ${M}; ${C}; ${W}; ${T}; ${B}; ${E}; ${H}; ${L}; ${S}; ${
     R}; ${N}; ${winners}; ${resources}; ${limits}; ${error}; ${leads}; ${
     verifications}; ${MAXCOV}; ${Clog2}; ${Mlog2}; ${Tlog2}; $resources;
     $cput; $mem; $vmem; $walltime1; $limits; $neednodes; $ppn1; $nodes;
     $ppn2; $walltime2; $outputDensityLines; $outputDensityChars; ${o_file
     }; ${e_file}; ${cfile}; ${ofile}; ${sfile}; ${lfile}; ${wfile}; ${
     grepper}; ${verified};"
echo XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXxx
##  echo "{notes}; {M}; {C}; {W}; {T}; {B}; {E}; {H}; {L}; {S}; {R}; {N}; {
   winners}; {resources}; {limits}; {error}; {leads}; {verifications}; {
   verified}; {MAXCOV}; {Clog2}; {Mlog2}; {Tlog2}; {grepper}"
  #echo "${notes}; ${M}; ${C}; ${W}; ${T}; ${B}; ${E}; ${H}; ${L}; ${S}; $
     {R}; ${N}; ${winners}; " >> tmt_results_table.csv
done
#http://stackoverflow.com/questions/918886/how-do-i-split-a-string-on-a-
   delimiter-in-bash
#https://ss64.com/bash/tr.html
exit
## c file, sh file, obj file, joblog file, e file, o file, win file
```

B.4 Other useful shell information and commands

- Show the list and status of queued, running, or freshly cancelled jobs:

```
qstat -u user <username>
```

- Show the list and status of queued, running, or freshly cancelled jobs:

```
canceljob <jobname>
```

Example:

```
canceljob 45553464.gpc-sched-ib0
```

Appendix C

Source Code

C.1 The main time-memory trade-off C program

```
tmt.c
/*
 * module load intel/15.0.2
 * icc hw.c -lssl -lcrypto
 * ./a.out
 * clear; fn=tmt07; rm *.log ; rm ./$fn.out; module load intel/15.0.2
    openmpi/intel/1.6.4; mpicc $fn.c -o $fn.out -lssl -lcrypto; date;
    mpirun -np 121 $fn.out ; date
 *
 * chmod +x tmt08.sh
 * qsub ./tmt08.sh
 * qstat 37228336.gpc-sched-ib0
 * showq
 *
 * gpc-f102n004-ib0-$ clear; fn=tmt17; rm *.log ; rm ./$fn.out; module
    load intel/15.0.2 openmpi/intel/1.6.4; mpicc $fn.c -o $fn.out -lssl -
    lcrypto; date; mpirun -np 5 $fn.out
 */
#include <stdio.h>
#include <sys/sysinfo.h>
#include <unistd.h>
```

```
#include <string.h>
#include <openssl/des.h>
#include "mpi.h"
#define MPI_MAX_LIBRARY_VERSION_STRING 5
#include <math.h>            /* pow */
#include <stdlib.h>
#include <time.h>
#include <float.h>
#include <stdint.h>

#include "simeck32.h"
#include "simeck48.h"
#include "simeck64.h"
#include "simeck32_32.h"
//#include "speck.cpp"

/*User must manually define those parameters*/
#define Total_M      780000 //65536+overhead 2^16
#define T 4
#define CipherUnderAttack    0      //XOR
#define BlockSize    16     //in bits
#define HalfBlock    0


//static
int test_confusion_once(void);
int test_diffusion_once(void);
int test_test_vector(void);

unsigned char *clear;
unsigned char *CTchallenge;
unsigned char **CTchallengeArray;
unsigned char *ReducedCTC;
unsigned char **ReducedCTCArray;
unsigned char *ChallengeKey;
unsigned char **ChallengeKeyArray;
int debug = 1; //0=none; 1=some, 2=comparisons and visual confirmation, 4=
    commented old printfs
```

```
/****************************************/
unsigned char *
Encrypt (unsigned char *Key, unsigned char *Msg, int size, unsigned char *
   Res)
{
  if (CipherUnderAttack == 1)
    {
      /*DES*/
      ////--static char *Res;
      int n = 0;
      DES_cblock Key2;
      DES_key_schedule schedule;
      ////--Res = (char *) malloc (size);
      /* Prepare the key for use with DES_cfb64_encrypt */
      memcpy (Key2, Key, 8);
      DES_set_odd_parity (&Key2);
      DES_set_key_checked (&Key2, &schedule);
      /* Encryption occurs here */
      DES_cfb64_encrypt ((unsigned char *) Msg, (unsigned char *) Res,
                         size, &schedule, &Key2, &n, DES_ENCRYPT);
      return (Res);
    }
  else if (CipherUnderAttack == 0)
    {
      /*XOR*/
        /*http://stackoverflow.com/questions/20579363/how-to-decrypt-simple
           -xor-encryption */
      int i, string_length = size;   //strlen(Msg);
      ////--static char *Res;
      ////--Res = (char *) malloc (string_length);
      for (i = 0; i < string_length; i++)
        {
          Res[i] = Msg[i] ^ Key[i];
          if (debug==3) printf(" xor=%i= ", Res[i]);
        }
      return (Res);
      ////--free (Res);
    }
```

```
  else if (CipherUnderAttack == 2) /*SIMECK */
    {
    if (size == 6){

    uint32_t PTi[] = {0,0,};
    uint32_t KYi[] = {0,0,0,0,};
    uint32_t CTi[] = {0,0,};
    PTi[0] = ( Msg[2] << 16 ) + ( Msg[1] <<8 ) + ( Msg[0]);
    PTi[1] = ( Msg[5] << 16 ) + ( Msg[4] <<8 ) + ( Msg[3]);

    KYi[0] = ( Key[2] << 16 ) + ( Key[1] <<8 ) + ( Key[0]);
    KYi[1] = ( Key[5] << 16 ) + ( Key[4] <<8 ) + ( Key[3]);

    //KYi[2] = ( KYc[8] << 16 ) + ( KYc[7] <<8 ) + ( KYc[6]);
    //KYi[3] = ( KYc[11] << 16 ) + ( KYc[10] <<8 ) + ( KYc[9]);
    simeck_48_96(KYi, PTi, CTi);

    Res[2] = CTi[0] >> 16 ;
    Res[1] = CTi[0] >> 8 ;
    Res[0] = CTi[0] & 0xFFFF;
    Res[5] = CTi[1] >> 16 ;
    Res[4] = CTi[1] >> 8 ;
    Res[3] = CTi[1] & 0xFFFF;


/*
     if (debug==2) printf ("sillybilly=== inKey is:::");
     char * verifyCTCptrx = Key ; int sillycounterx=0;
     if (debug==2) for (sillycounterx=0; sillycounterx<size; sillycounterx
        ++){ printf("[%d_%02x]",sillycounterx, *verifyCTCptrx);
        verifyCTCptrx++;}
*/
/*
     uint32_t * ResInt = calloc(2, sizeof(uint32_t));
     uint32_t * KeyInt = calloc(2, sizeof(uint32_t));
     uint32_t * MsgInt = calloc(2, sizeof(uint32_t));
     memcpy (KeyInt, Key, (size ));
     memcpy (MsgInt, Msg, (size) );
```

```
     simeck_48_96 (KeyInt, MsgInt, ResInt);
     memcpy (Res, ResInt, (size) );
*/

/*     uint32_t Res_integered[] = { 0, 0, };
     uint32_t Key_integered[] = { 0, 0, };
     Key_integered[0] = (Key[2] << 16) + (Key[1] << 8) + Key[0];
     Key_integered[1] = (Key[5] << 16) + (Key[4] << 8) + Key[3];

     uint32_t Msg_integered[] = { 0, 0, };
     Msg_integered[0] = (Msg[2] << 16) + (Msg[1] << 8) + Msg[0];
     Msg_integered[1] = (Msg[5] << 16) + (Msg[4] << 8) + Msg[3];

     simeck_48_96 (Key_integered, Msg_integered, Res_integered);
     Res[0] = Res_integered[0] & 0xFF;
     Res[1] = Res_integered[0] >> 8;
     Res[2] = Res_integered[0] >> 16;
     Res[3] = Res_integered[1] & 0xFF;
     Res[4] = Res_integered[1] >> 8;
     Res[5] = Res_integered[1] >> 16;

     if (debug==2) printf (" === ct is:");
     verifyCTCptrx = Res_integered ; // warning #556: a value of type "
        uint32_t={unsigned int} *" cannot be assigned to an entity of
        type "char *"
     if (debug==2) for (sillycounterx=0; sillycounterx<size; sillycounterx
        ++){ printf("[%d_%02x]",sillycounterx, *verifyCTCptrx);
        verifyCTCptrx++;}
     if (debug==2) printf (" === PT is:"); // warning #556: a value of
        type "uint32_t={unsigned int} *" cannot be assigned to an entity
        of type "char *"
     verifyCTCptrx = Msg_integered;
     if (debug==2) for (sillycounterx=0; sillycounterx<size; sillycounterx
        ++){ printf("[%d_%02x]",sillycounterx, *verifyCTCptrx);
        verifyCTCptrx++;}
     if (debug==2)  printf("\n");
```

```
     if (debug==2) printf("sillybilly ENC ResInt: %08x %08x \t KeyInt: %08
        x %08x OrigPT: %08x %08x \n", Res_integered[0], Res_integered[1],
         Key_integered[0], Key_integered[1], Msg_integered[0],
        Msg_integered[1] );
*/


     return (Res);

   } else if (size == 4){
     ////--static char *Res;
     ////--Res = (char *) malloc (size);
     uint16_t Res_integered[] = { 0, 0, };
     /* http://stackoverflow.com/questions/27558956/how-to-convert-a-char-
        array-to-a-uint16-t-by-casting-type-pointer */
     uint16_t Key_integered[2];
     Key_integered[0] = (Key[1] << 8) + Key[0];
     Key_integered[1] = (Key[3] << 8) + Key[2];
     /*Key_integered[2] = (Key[5] << 8) + Key[4];
       Key_integered[3] = (Key[7] << 8) + Key[6];
      */
     uint16_t Msg_integered[2];
     Msg_integered[0] = (Msg[1] << 8) + Msg[0];
     Msg_integered[1] = (Msg[3] << 8) + Msg[2];
     /* simeck_32_64(key64, text32, CT32); */
     /*simeck_32_64_Dec(key64_Dec, CT32, PT32_Dec); */
     //simeck_32_32 (Key_integered, Msg_integered, Res_integered);
     simeck_32_64 (Key_integered, Msg_integered, Res_integered);
/*   simeck_32_32_Dec(key32_32_Dec, CT32, PT32_Dec); */
/* simeck_32_64(Key_integered, Msg_integered, Res_integered); */
     /* http://stackoverflow.com/questions/13279024/convert-a-uint16-t-to-
        char2-to-be-sent-over-socket-unix */
     Res[0] = Res_integered[0] & 0xFF;
     Res[1] = Res_integered[0] >> 8;
     Res[2] = Res_integered[1] & 0xFF;
     Res[3] = Res_integered[1] >> 8;
     return (Res);
     ////--free (Res);
```

```
    }/// else if (size ==6 ){}

  }

}

/****************************************/
unsigned char *
Decrypt (unsigned char *Key, unsigned char *Msg, int size, unsigned char *
   Res)
{
  if (CipherUnderAttack == 1)
    {
      /*DES*/ static char *Res;
      int n = 0;
      DES_cblock Key2;
      DES_key_schedule schedule;
      Res = (char *) malloc (size);
      /* Prepare the key for use with DES_cfb64_encrypt */
      memcpy (Key2, Key, 8);
      DES_set_odd_parity (&Key2);
      DES_set_key_checked (&Key2, &schedule);
      /* Decryption occurs here */
      DES_cfb64_encrypt ((unsigned char *) Msg, (unsigned char *) Res,
                         size, &schedule, &Key2, &n, DES_DECRYPT);
      return (Res);
    }
  else if (CipherUnderAttack == 0)
    {
      /*XOR*/
        /*http://stackeverflow.com/questions/20579363/how-to-decrypt-simple
           -xor-encryption */
      int i, string_length = size;   //strlen(Msg);
      static char *Res;
      Res = (char *) malloc (string_length);
      for (i = 0; i < string_length; i++)
        {
          Res[i] = Msg[i] ^ Key[i];
```

```
        if (debug==3) printf("%i", Res[i]);
      }
    return (Res);
    //free (Res);
  }
else if (CipherUnderAttack == 2)
  {                          /*Simeck */
     if (size == 4){
    //static char *Res;
    uint16_t Res_integered[] = { 0, 0, };
    //Res = (char *) malloc (size);
    /* http://stackoverflow.com/questions/27558956/how-to-convert-a-char-
       array-to-a-uint16-t-by-casting-type-pointer */
    uint16_t Key_integered[2];
    Key_integered[0] = (Key[1] << 8) + Key[0];
    Key_integered[1] = (Key[3] << 8) + Key[2];
    uint16_t Msg_integered[2];
    Msg_integered[0] = (Msg[1] << 8) + Msg[0];
    Msg_integered[1] = (Msg[3] << 8) + Msg[2];
    //simeck_32_32_Dec (Key_integered, Msg_integered, Res_integered);
    simeck_32_64_Dec (Key_integered, Msg_integered, Res_integered);
    /* http://stackoverflow.com/questions/13279024/convert-a-uint16-t-to-
       char2-to-be-sent-over-socket-unix */
    Res[0] = Res_integered[0] & 0xFF;
    Res[1] = Res_integered[0] >> 8;
    Res[2] = Res_integered[1] & 0xFF;
    Res[3] = Res_integered[1] >> 8;
    return (Res);
    //free (Res);
      } else if (size == 6){


  uint32_t RTi[] = {0,0,};
  //uint32_t PTi[] = {0,0,};
  uint32_t KYi[] = {0,0,0,0,};
  uint32_t CTi[] = {0,0,};
  CTi[0] = ( Msg[2] << 16 ) + ( Msg[1] <<8 ) + ( Msg[0]);
  CTi[1] = ( Msg[5] << 16 ) + ( Msg[4] <<8 ) + ( Msg[3]);
```

```
    KYi[0] = ( Key[2] << 16 ) + ( Key[1] <<8 ) + ( Key[0]);
    KYi[1] = ( Key[5] << 16 ) + ( Key[4] <<8 ) + ( Key[3]);

    //KYi[2] = ( KYc[8] << 16 ) + ( KYc[7] <<8 ) + ( KYc[6]);
    //KYi[3] = ( KYc[11] << 16 ) + ( KYc[10] <<8 ) + ( KYc[9]);



    simeck_48_96_Dec(KYi, CTi, RTi);

    Res[2] = RTi[0] >> 16 ;
    Res[1] = RTi[0] >> 8 ;
    Res[0] = RTi[0] & 0xFFFF;
    Res[5] = RTi[1] >> 16 ;
    Res[4] = RTi[1] >> 8 ;
    Res[3] = RTi[1] & 0xFFFF;


/*
      uint32_t * ResInt = calloc(2, sizeof(uint32_t));
      uint32_t * KeyInt = calloc(2, sizeof(uint32_t));
      uint32_t * MsgInt = calloc(2, sizeof(uint32_t));
      memcpy (KeyInt, Key, (size)) ;
      memcpy (MsgInt, Msg, (size)) ;
      simeck_48_96_Dec (KeyInt, MsgInt, ResInt);
      memcpy (Res, ResInt, (size) );
      printf("sillybilly Key: %08x %08x \t KeyInt: %08x %08x \n", Key[0],
         Key[1], KeyInt[0], KeyInt[1] );
*/
/*
      uint32_t Res_integered[] = { 0, 0, };
      uint32_t Key_integered[] = { 0, 0, };
      Key_integered[0] = (Key[2] << 16) + (Key[1] << 8) + Key[0];
      Key_integered[1] = (Key[5] << 16) + (Key[4] << 8) + Key[3];

      uint32_t Msg_integered[] = { 0, 0, };
      Msg_integered[0] = (Msg[2] << 16) + (Msg[1] << 8) + Msg[0];
```

```
      Msg_integered[1] = (Msg[5] << 16) + (Msg[4] << 8) + Msg[3];

      simeck_48_96_Dec (Key_integered, Msg_integered, Res_integered);
      Res[0] = Res_integered[0] & 0xFF;
      Res[1] = Res_integered[0] >> 8;
      Res[2] = Res_integered[0] >> 16;
      Res[3] = Res_integered[1] & 0xFF;
      Res[4] = Res_integered[1] >> 8;
      Res[5] = Res_integered[1] >> 16;

      if (debug==2) printf("sillybilly DEC MsgInt : %08x %08x \t KeyInt:
         %08x %08x Recovered:: %08x %08x \n", Msg_integered[0],
         Msg_integered[1], Key_integered[0], Key_integered[1] ,
         Res_integered[0], Res_integered[1] );
*/


      return (Res);
        }
    }

}

/***************************************/
int
CalculateEndPoint (unsigned char *SPp, unsigned char *EPp, int BS, int
   record, int taskid,
                double BruteForceNumberOfComputations, double M,
                double WallTime, int TT)
{

  int t = 0;
  unsigned char *CipherT;
  CipherT = calloc (BS, sizeof (char));
  unsigned char *key;
  key = malloc (BS);
  memcpy (key, SPp, BS);
  for (t = 0; t < TT; t++)
```

```
    {
if (debug==3) {             printf("\n\nBefore taskid=%d T=%d record=%d t
   =%d ",taskid, TT, record , t);
               int b=0;
               printf("Cipher/EP:\t");
               for (b=0; b< BS; b++){
                      printf("_%c",CipherT[b]);
               }
               printf("\tPTxt\t");
               for (b=0; b< BS; b++){
                      printf("_%c",clear[b]);
               }
               printf("\tKey/SP:\t");
               unsigned char * keyp = key;
               for (b=0; b< BS; b++){
                      printf("_%c",*keyp);
                      keyp++;
               }
               printf("\t\n");
}
     if (debug==3) printf("\t%d", t);
     unsigned char *Res = (unsigned char *) malloc (BS);
     memcpy (CipherT, Encrypt (key, clear, BS, Res), BS);
     free (Res);
     //printf("    3.x memcpy .....\n");
     //key = CipherT; /// this used to work because they were not
        sepatrate memory address
//////////memcpy (key, CipherT, BS); // this one actually moves the value.
     //printf("T=%d record=%d t=%d CT==%04x  %04x=     Ky==%04x    %04x=\n
        ",TT, record , t, CipherT[0],CipherT[1], key[0], key[1]);

if (debug==3) {
               if (taskid ==1){
               printf("\n\nAfterr taskid=%d T=%d record=%d t=%d ",taskid,
                  TT, record , t);
               unsigned char * CTp = CipherT;
               int b=0;
               printf("Cipher/EP:\t");
```

```
                for (b=0; b< BS; b++){
                        printf("_%02x", *CTp);
                        CTp++;
                }
                printf("\tPTxt\t");
                for (b=0; b< BS; b++){
                        printf("_%02x",clear[b]);
                }
                printf("\tKey:\t");
                unsigned char * keyp = key;
                b=0;
                for (b=0; b< BS; b++){
                        printf("_%02x",*keyp);
                        keyp++;
                }
                printf("\tSP:\t");
                unsigned char * sppp = SPp;
                for (b=0; b< BS; b++){
                        printf("_%02x",*sppp);
                        sppp++;
                }
                printf("\t\n");
                }
}
memcpy (key, CipherT, BS);   // this one actually moves the value.

    }

  //printf("\ncore:%d DONE LOOP record=%d .....\n",taskid, record);
  //printf(".....\n");
  //memcpy(&EP[record+0], CipherT, BS);
  memcpy (EPp, CipherT, BS);

  free (CipherT);
  free (key);

  return (0);
}
```

```
/**************************************/
int
RegenerateKeyndPoint (unsigned char *SPp, unsigned char *EPp, int BS, int
   record, int taskid,
                   unsigned char *RecoveredKey, int TT)
{
  int t = 0;
  unsigned char *CipherT;
  CipherT = malloc (BS);
  unsigned char *key;
  key = malloc (BS);
  unsigned char *SPpa = SPp;
  memcpy (key, SPpa, BS);
  memcpy (CipherT, SPpa, BS);
      unsigned char *Res = (unsigned char *) malloc (BS);
      //memcpy (CipherT, Encrypt (key, clear, BS, Res), BS);
      //free (Res);
  for (t = 0; t < TT; t++)
    {
      //printf("%d",t);
      memcpy (CipherT, Encrypt (key, clear, BS, Res), BS);
      memcpy (key, CipherT, BS);
    }
     free(Res);
  memcpy (RecoveredKey, CipherT, BS);
  unsigned char *rkaddr = RecoveredKey;
  int c = 0;
  printf ("\tRcoveredKey is:::::: ");
  for (c = 0; c < BS; c++)
    {
      printf ("___%c__", *rkaddr);
      rkaddr++;
    }
  free (CipherT);
  free (key);
  return (0);
}
```

```
/****************************************/
/* http://stackoverflow.com/questions/699968/display-the-binary-
   representation-of-a-number-in-c
 * Create a string of binary digits based on the input value.
 *  Input:
 *  val: value to convert.
 *  buff: buffer to write to must be >= sz+1 chars.
 *  sz:   size of buffer.
 *  Returns address of string or NULL if not enough space provided.
 * */
static char *
binrep (unsigned int val, char *buff, int sz)
{
  char *pbuff = buff;
  /* Must be able to store one character at least. */
  if (sz < 1)
    return NULL;
  /* Special case for zero to ensure some output. */
  if (val == 0)
    {
      *pbuff++ = '0';
      *pbuff = '\0';
      return buff;
    }
  /* Work from the end of the buffer back. */
  pbuff += sz;
  *pbuff-- = '\0';
  /* For each bit (going backwards) store character. */
  while (val != 0)
    {
      if (sz-- == 0)
        return NULL;
      *pbuff-- = ((val & 1) == 1) ? '1' : '0';
      /* Get next bit. */
      val >>= 1;
    }
  return pbuff + 1;
```

```
}

/*******************************************************/
int
searchEPforCT (unsigned char *SPp, unsigned char *EPp, int rank, int
   offset, int recordQ,
              unsigned char *reduced_, unsigned char *encrypted_, int t,
                 double BlockSizeBytes,
              unsigned char *OriginalKey, unsigned char *OriginalCT) //
                 OriginalCT is different from encrypted because it doesnt
                 change (reencrypt)everytime
{
  printf("\nsearchEPforCT rank %d, t: %d ------- offset: %d", rank, t,
     offset );
  /*http://www.c4learn.com/c-programming/c-passing-array-of-structure-to-
     function/ */
  if (t < 0)
    return (-1);
  int record = 0;
  int i = 0;
  int lead = 0;
  int done = 0;
  int finished =0;
  /* http://en.cppreference.com/w/c/string/byte/strcmp */
  for (i = 0; i < recordQ; i = i + 1)
    {
      int cntr = 0;
      int same = 0;
      unsigned char *EPpa = &EPp[i];
      unsigned char *CTpa = encrypted_;
      for (cntr = 0; cntr < BlockSizeBytes; cntr++)
        {
        if (debug==2) printf ("\n@@@@@ Rank %d , Record %d T %d --
           Comparing # %d CT byte [%02x]&[%02x] the EP byte \n", rank, i, t
           , cntr, *CTpa, *EPpa);
          if (*CTpa == *EPpa)
            {
```

```
          //printf ("\n@@@@@ Rank %d , Record %d T %d -- Comparing # %d
              CT byte [%02x]&[%02x] the EP byte \n", rank, i, t, cntr,
              *CTpa, *EPpa);
          same = same + 1;
          if (same > 2)
            {
              if (debug==2)
                printf ("\n @@@@@ Rank %d , Record %d T %d-- Yaaaa %d
                   similar blocks up to %d- %c _ %02x = %c _%02x \n",
                 rank, i, t, same, cntr, *CTpa, *EPpa, *CTpa, *EPpa);
            }
          CTpa++;
          EPpa++;
        }
      else
        {
          same = 0;
          //printf("----------------------outta here same is %d\n",
             same);
          break;
        }
    }
  if (same == BlockSizeBytes)
    {                     // BlockSizeBytes)
      if (debug >= 2) printf
        ("\n\nX>>>>>>>>>>>>> inside search : Yaaaay We have a lead t=%d
            core=%d m=%d . BS is %d and same_bytes are %d ",
         t, rank, i, BlockSizeBytes, same);
      record = i;
      lead = 1;
      /*Now i need to extract the key by Calculating from SPi T-1 */
      unsigned char *RecoveredKey;
      RecoveredKey = malloc (BlockSizeBytes);
      unsigned char *SPppp = &SPp[i];
      //printf("\nxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx>>>> %
         c%c%c%c \n",*SPppp++, *SPppp++, *SPppp++, *SPppp);
      int result =
        RegenerateKeyndPoint (&SPp[i], &EPp[i], BlockSizeBytes, record,
```

```
                        rank, RecoveredKey, t - 1);
printf ("\nYaaaay lead at t=%d core=%d, m=%d, key1stB_%c_",
       t, rank, i, *RecoveredKey);

int b = 0;
printf (" Orig PT:");
for (b = 0; b < BlockSizeBytes; b++)
  {
    printf ("_[%c][%02x]_", clear[b], clear[b]);
  }

printf (" Found CT:");
unsigned char *eppp = EPp;;
for (b = 0; b < BlockSizeBytes; b++)
  {
    printf ("_[%c][%02x]_", *eppp, *eppp);
    eppp++;
  }

printf (" Recovered Key(EP-1):");
unsigned char *keyp = RecoveredKey;
for (b = 0; b < BlockSizeBytes; b++)
  {
    printf ("_[%c][%02x]_", *keyp, *keyp);
    keyp++;
  }
printf (" Chain SP:");
unsigned char *sppp = SPp;
for (b = 0; b < BlockSizeBytes; b++)
  {
    printf ("_[%c][%02x]_", *sppp, *sppp);
    sppp++;
  }
printf (" Orig CT:");
unsigned char *octp = OriginalCT;
for (b = 0; b < BlockSizeBytes; b++)
  {
    printf ("_[%c][%02x]_", *octp, *octp);
```

```
    octp++;
  }
printf (" Orig Key:");
unsigned char *okyp = OriginalKey;
for (b = 0; b < BlockSizeBytes; b++)
  {
    printf ("_[%c][%02x]_", *okyp, *okyp);
    okyp++;
  }
printf ("  CT 4 verify:");
unsigned char *verifyCT;
verifyCT = malloc (BlockSizeBytes);
unsigned char *__result= malloc (BlockSizeBytes);
memcpy (verifyCT, Encrypt (RecoveredKey, clear, BlockSizeBytes,
   __result),
       BlockSizeBytes);
free(__result);
unsigned char * vctp = verifyCT;
for (b = 0; b < BlockSizeBytes; b++)
  {
    printf ("_[%c][%02x]_", *vctp, *vctp);
    vctp++;
  }

vctp = verifyCT;
octp = OriginalCT;
int bytesVerified=0;
for (b = 0; b < BlockSizeBytes; b++)
  {
         //printf(" Verifying... [%02x][%02x]..", *vctp, *octp);
     if (*vctp++ == *octp++){
         bytesVerified++;
         if (bytesVerified==BlockSizeBytes){
             printf("Yaaaay: We have Verification also.");
         }
     }else{
         bytesVerified=0;
     }
```

```
          }
        free (verifyCT);
        printf ("\n");
        free (RecoveredKey);
        //done = 1;
        //break;
      }
    else
      {
      }
  }
/*What if we couldn't find it in the EP's of T. We need to do it for T-1
    */
if (done == 1) // && finished == 1)
  {
    return (record);
  }
else
  {
    unsigned char *Y2CTC;
    unsigned char *ReducedCTC2;
    ReducedCTC2 = malloc (BlockSizeBytes);
    Y2CTC = malloc (BlockSizeBytes);
    unsigned char *Res = (unsigned char *) malloc (BlockSizeBytes);
    //memcpy (CipherT, Encrypt (key, clear, BS, Res), BS);
    //free (Res);
    memcpy (Y2CTC, Encrypt (encrypted_, clear, BlockSizeBytes, Res),
           BlockSizeBytes);
    free (Res);
    //ReducedCTC2 = malloc (BlockSizeBytes);
    record =
      searchEPforCT (SPp, EPp, rank, offset, recordQ, ReducedCTC2, Y2CTC,
                     t - 1, BlockSizeBytes, OriginalKey, OriginalCT);
    free (ReducedCTC2);
    free (Y2CTC);

  }
return record;
```

```
}

/*********************************************************/
/*********************************************************/
double
testEncDec (int rank, int BlockSizeBytes, unsigned char *TestKy, unsigned
   char *TestCT,
           unsigned char *TestPT, unsigned char *TestRT)
{
  /* http://stackoverflow.com/questions/9655202/how-to-convert-integer-to-
     string-in-c */
  /* http://www.cplusplus.com/reference/cstring/strcat */
  clock_t start, end;
  double cpu_time_used;
  start = clock ();
  int b = 0;
  for (b = 0; b < BlockSizeBytes; b++)
    {
      TestKy[b] = 'K';
      TestPT[b] = 'p';
    }
  int x = 0;
  printf ("\n\ttestEncDec: Yaaa Test Key\t : ");
  for (b = 0; b < BlockSizeBytes; b++)
    {
      printf ("_%c_", TestKy[b]);
    }
  printf ("\n\ttestEncDec: Yaaa Clear text\t : ");
  for (b = 0; b < BlockSizeBytes; b++)
    {
      printf ("_%c_", TestPT[b]);
    }
      unsigned char *Res = (unsigned char *) malloc (BlockSizeBytes);
      //memcpy (CipherT, Encrypt (key, clear, BS, Res), BS);
      //free (Res);
  memcpy (TestCT, Encrypt (TestKy, TestPT, BlockSizeBytes, Res),
     BlockSizeBytes);
      free (Res);
```

```
  printf ("\n\ttestEncDec: Yaaa Encrypted text\t : ");
  for (b = 0; b < BlockSizeBytes; b++)
    {
      printf ("_%c_", TestCT[b]);
    }
  unsigned char * ___result= malloc (BlockSizeBytes);
  memcpy (TestRT, Decrypt (TestKy, TestCT, BlockSizeBytes, ___result),
     BlockSizeBytes);
  free(___result);
  printf ("\n\ttestEncDec: Yaaa Decrypted text\t : ");
  for (b = 0; b < BlockSizeBytes; b++)
    {
      printf ("_%c_", TestRT[b]);
    }
  sleep (10);
  end = clock ();
  cpu_time_used = ((float) (end - start)) / CLOCKS_PER_SEC;
  printf ("\n\ttestEncDec: Yaaa Test time\t : %G \n\n\n ", cpu_time_used);

  return cpu_time_used;
}

/*************************************************************/

int
calculateLFSR (int BlockSizeBits, unsigned char **LFSRSP, int rank,
   unsigned char **SP,
              int recordQuota, uint32_t lfsrNOTNEEDED)
{
  if (BlockSizeBits == 32)
    {
      uint32_t bit32;
      uint32_t lfsr32;
      int BlockSizeBytes = BlockSizeBits / 8;
      printf ("\nrank %d before memcpy
         ------------------------------------",
              rank);
      memcpy (&lfsr32, LFSRSP[rank - 1], BlockSizeBytes);
```

```
     printf ("\nrank %d after memcpy----------------------------------",
            rank);
     int period = 0;
     int lfsrspcount = 0;
     /* https://en.wikipedia.org/wiki/Linear-feedback_shift_register */
     do
       {
         /*printf ("\n %d",period);*/
         unsigned char lo = lfsr32 & 0xFF;
         unsigned char b1 = lfsr32 >> 8;
         unsigned char b2 = lfsr32 >> 16;
         unsigned char hi = lfsr32 >> 24;
         memcpy (SP[period], &lfsr32, BlockSizeBytes);

/*         char *adrss= SP[period];
        printf("\n>>>>>>>>>>>>>>>>>>FINDME.....Child Process %d inside
           Lfsr value in SP is: ", rank);
        int bbb=0;
        for (bbb=0; bbb<BlockSizeBytes; bbb++){ printf("%d_%c ",bbb, *
           adrss); adrss++;}
        printf("\n");
*/

         period = period + 1; //BlockSizeBytes;
         /*primitive polynomial from Kali: y^32 + y^15 + y^9 +       y^7 +
             y^4 +                  y^3 +                        1
            tap sequence [32, 7, 3, 2, 0]
            wikipedia polynomial=       x^32  +                     x^7
                                  +   X^3                 +    X^2  + X^0
                */
         bit32 =
           ((lfsr32 >> (BlockSizeBits - 32)) ^
            (lfsr32 >> (BlockSizeBits - 7)) ^ (lfsr32 >> (BlockSizeBits -
               3))
            ^ (lfsr32 >> (BlockSizeBits - 2))) & 1;
         lfsr32 = (lfsr32 >> 1) | (bit32 << (BlockSizeBits - 1));
       }
     while (period < recordQuota);
```

```
    printf ("M======= core %d just LFSRd %d times. >>>>>\n", rank, period
       );
  }
 else if (BlockSize == 48)
      {
      uint64_t lfsr64 = 0x0000555555FF0000u;

        uint16_t lfsr48[]={ 0x0123, 0x4567, 0x89ab };
        uint64_t bit64;
    int BlockSizeBytes = BlockSizeBits / 8;
    printf ("\nrank %d before memcpy
       ------------------------------------",
           rank);
    memcpy (&lfsr64, LFSRSP[rank - 1], BlockSizeBytes);
    printf ("\nrank %d after memcpy----------------------------------",
           rank);
    int period = 0;
    int lfsrspcount = 0;
    /* https://en.wikipedia.org/wiki/Linear-feedback_shift_register */
    do
      {
        /*printf ("\n %d",period);
        char lo = lfsr48 & 0xFF;
        char b1 = lfsr48 >> 8;
        char b2 = lfsr48 >> 16;
        char h3 = lfsr48 >> 24;
        char b4 = lfsr48 >> 32;
        char hi = lfsr48 >> 40;*/

        memcpy (SP[period], &lfsr64, BlockSizeBytes);

        period = period + 1; //BlockSizeBytes;
        /*primitive polynomial from Kali: y^32 + y^15 + y^9 +       y^7 +
            y^4 +                  y^3 +                        1
*            tap sequence [32, 7, 3, 2, 0]
*                        wikipedia polynomial=       x^32  +
                        x^7                     +   X^3                 +
       X^2  + X^0 */
```

```
        bit64 =
          (( lfsr64 >> (BlockSizeBits - 32)) ^
           ( lfsr64 >> (BlockSizeBits - 7)) ^
           ( lfsr64 >> (BlockSizeBits - 3)) ^
           ( lfsr64 >> (BlockSizeBits - 2))) & 1;
        lfsr64 = ( lfsr64 >> 1) | (bit64 << (BlockSizeBits - 1));
       }
     while (period < recordQuota);
     printf ("M======= core %d just LFSRd %d times. >>>>>\n", rank, period
        );

       }

}


/*http://www.linux-mag.com/id/1332/*/
int
main (int argc, char **argv)
{
  clock_t just_started, before_lfsr_seed, before_dispatch, before_results,
    just_finished;
  clock_t awaiting_dispatch, before_randomizing, before_calculating,
    before_sorting, before_searching, before_reporting;
  double total_time, master_time_to_seed;
  double child_time_to_randomize, child_time_to_calculate,
    child_time_to_search;
  just_started = clock ();
     //printf ("\nA============================ 0 \n");

  int skipSort = 0;
  unsigned long long BlockSizeBits = BlockSize; // 16 bits key and block
     size
  unsigned long long BlockSizeBytes = BlockSizeBits / 8;  // In bytes
  clear = malloc (BlockSizeBytes);
/*          8 32  64    128    */
  unsigned char st[] = "PTPTPTPTPTPTPTPT";
     //printf ("\nA============================ 0.1 \n");
```

```
memcpy (clear, st, BlockSizeBytes);
CTchallenge = malloc (BlockSizeBytes);
CTchallengeArray = malloc (BlockSizeBytes * 100);
ChallengeKey = malloc (BlockSizeBytes);
ChallengeKeyArray = malloc (BlockSizeBytes * 100);


unsigned char ck[] = "KCKCKCKCKCKCKCKC";
    //printf ("\nA============================ 0.3 \n");
memcpy (ChallengeKey, ck, BlockSizeBytes);
    unsigned char *Res = (unsigned char *) malloc (BlockSizeBytes);

    printf
      ("C sillybilly ========================== /** / and INkey is: ");
    unsigned char * v_ = ChallengeKey ; int s_=0;
    for (s_=0; s_<BlockSizeBytes; s_++){ printf("[%d_%02x]",s_, *v_); v_
       ++;}
      printf(" \n");



    //memcpy (CipherT, Encrypt (key, clear, BS, Res), BS);
    //free (Res);
    //printf ("\nA============================ 0.4 \n");
memcpy (CTchallenge, Encrypt (ChallengeKey, clear, BlockSizeBytes, Res),
       BlockSizeBytes);
    free (Res);

    printf
      ("C silly ========================== /** / \t\t and CT forChallenge
          is: ");
    unsigned char * verifyCTCptrx = CTchallenge ; int sillycounterx=0;
    for (sillycounterx=0; sillycounterx<BlockSizeBytes; sillycounterx++){
        printf("%d_%02x ",sillycounterx, *verifyCTCptrx); verifyCTCptrx
       ++;}
      printf("\n");

       unsigned char *verifyPT_;
```

```
        verifyPT_ = malloc (BlockSizeBytes);
        unsigned char * _result= malloc (BlockSizeBytes);
        memcpy (verifyPT_, Decrypt (ChallengeKey, CTchallenge,
           BlockSizeBytes, _result),
               BlockSizeBytes);
       free(_result);
        unsigned char * vptp_ = verifyPT_;
    printf ("\nsillybilly and when CT is decryoted it is back to : \t");
        int b_=0; for (b_ = 0; b_ < BlockSizeBytes; b_++)
          {
            printf ("_[%c][%02x]_", *vptp_, *vptp_);
            vptp_++;
          }
        printf ("\t\n");
      //assert (*vptp_ == *s_ );
        free (verifyPT_);



    //printf ("\nA============================ 0.5 \n");
  ReducedCTC = malloc (BlockSizeBytes);
    //printf ("\nA============================ 0.6 \n");
  uint32_t start_state32 = 0x434B434Bu;

  int ArraySizeInBytes = Total_M * BlockSizeBytes;
  //int V = ArraySizeInBytes / Cores;
  int i, rank, size, dest, src, tag;
  double requiredMemoryInMegaBytes = ArraySizeInBytes * 2 / 1024 / 1024;
  double memoryPerCoreInMegaBytes = 2 * 1024 ;
//     printf ("\nA============================ requires memory in
   megabytes = ArraySizeInBytes %d x2/1024/1024 = %d \n", ArraySizeInBytes
   , requiredMemoryMB);

    //printf ("\nA============================ 0.7 \n");


  int numbreOfChallenges =100;
  unsigned char **CTArray;
```

```
  unsigned char **KYArray;
  unsigned char PTArray[]= "KCKCKCKCKCKC Prof. Guang Gong, Ph.D., IEEE
     Fellow Department of Electrical and Computer Engineering Office: EIT
     4158 Tel: (519) 888-4567 x 35650 Fax: (519) 746-3077 Email: Homepage:
      http://comsec.uwaterloo.ca/~ggong/ (or access from ECE Dept.
     homepage) Prof. Mark Aagaard, Ph.D. Department of Electrical and
     Computer Engineering Office: DC 2539 Tel: (519) 888-4567 x 33138
     Email: Homepage: https://ece.uwaterloo.ca/~maagaard/ (or access from
     ECE Dept. homepage) Our Research Areas Lightweight cryptography,
     cryptography and elliptic curve cryptography Signal design for
     wireless CDMA, OFDM and MIMO communications Security in cloud,
     network, ad-hoc network and RFID systems Wireless and multimedia
     communication security Security and privacy in machine-to-machine
     communication";
  KYArray = (unsigned char **) calloc ( numbreOfChallenges, sizeof (
     unsigned char *));
  if (KYArray == NULL)
    {
      fprintf (stderr, "not enough memory for Challenge Array\n");
    }
  unsigned char *blobzz = malloc ( numbreOfChallenges * BlockSizeBytes *
     sizeof (unsigned char));
  if (blobzz == NULL)
    {
      fprintf (stderr, "not enough memory for Challenge Array Blob\n");
    }
  else
    {
       memcpy (blobzz, PTArray , BlockSizeBytes * numbreOfChallenges );

    }
  int iizz = 0;
  for (iizz = 0; iizz < numbreOfChallenges; iizz++)
    {
      KYArray[iizz] = &blobzz[iizz * BlockSizeBytes];
    }
/************/
```

```
    //printf ("\nA============================ 0.8 \n");

CTArray = (unsigned char **) calloc ( numbreOfChallenges, sizeof (
   unsigned char *));
if (CTArray == NULL)
  {
    fprintf (stderr, "not enough memory for Challenge Array\n");
  }
unsigned char *blobzzz = calloc ( numbreOfChallenges , BlockSizeBytes *
   sizeof (unsigned char));
if (blobzzz == NULL)
  {
    fprintf (stderr, "not enough memory for Challenge Array Blob\n");
  }
else{

  }
int iizzz = 0;
for (iizzz = 0; iizzz < numbreOfChallenges; iizzz++)
  {
     unsigned char * KYAp = (unsigned char *) KYArray[iizzz] ;

    unsigned char * Ress = (unsigned char *) malloc (BlockSizeBytes);
    CTArray[iizzz] = &blobzzz[iizzz * BlockSizeBytes];
    unsigned char * CTAp = (unsigned char *) CTArray[iizzz] ;
    memcpy ( CTAp, Encrypt ( KYAp, clear, BlockSizeBytes, Ress),
       BlockSizeBytes);


      unsigned char * ctp = CTArray[iizzz];
      unsigned char * ptp = clear;
      unsigned char * kyp = KYArray[iizzz];
      int bb=0;
      for (bb=0; bb<BlockSizeBytes; bb++){
             printf("\nkey %d count %d PTx[%02x] PTc[%c] KYx[%02x] KYc[%c
                ] CTx[%02x] CTc[] >>",iizzz, bb, *ptp, *ptp, *kyp, *kyp,
                 *ctp /*, *ctp*/);
```

```
            printf("\t... PTx[%02x] PTc[%c] KYx[%02x] KYc[%c] CTx[%02x]
               CTc[%c]", *ptp, *ptp, *KYAp, *KYAp, *CTAp, *CTAp);
            //printf("\n PTx[%02x] PTc[%c] KYx[%02x] KYc[%c] CTx[%02x]
               ", *ptp, *ptp, *kyp, *kyp, *ctp);
            //printf("\n PTx[%02x] PTc[%c] KYx[%02x] KYc[%c] ", *ptp, *
               ptp, *kyp, *kyp);
            //                                                  //printf("\n
               PTx[%02x] PTc[%c] KYx[%02x] ", *ptp, *ptp, *kyp);
            //
               //printf("\n PTx[%02x] PTc[%c] ", *ptp, *ptp );
            ctp++; kyp++; ptp++; CTAp++; KYAp++;
    }
   printf ("\nA============================== challange key %d \n", iizzz
      );
            //




  free (Ress);
}
/*printf("Number of challenges
   ========================================i %d\n",
   numbreOfChallenges);
 int sss=0;
 for (sss=0; sss<numbreOfChallenges; sss++){
    int bb=0;
    char * ctp = CTArray[sss];
    char * ptp = clear;
    char * kyp = KYArray[sss];
    for (bb=0; bb<BlockSizeBytes; bb++){
            //printf("\n PTx[%02x] PTc[%c] KYx[%02x] KYc[%c] CTx[%02x]
               CTc[%c]", *ptp, *ptp, *kyp, *kyp, *ctp, *ctp);
            //printf("\n PTx[%02x] PTc[%c] KYx[%02x] KYc[%c] CTx[%02x]
               ", *ptp, *ptp, *kyp, *kyp, *ctp);
            printf("\n PTx[%02x] PTc[%c] KYx[%02x] KYc[%c] ", *ptp, *ptp
               , *kyp, *kyp);
```

```
              //printf("\n PTx[%02x] PTc[%c] KYx[%02x] ", *ptp, *ptp, *kyp
                 );
              //printf("\n PTx[%02x] PTc[%c] ", *ptp, *ptp );
              ctp++; kyp++; ptp++;
       }
       printf("xxxx..................................... %d <==>\t %c \t
          <==> %c \n",sss,CTArray[sss], KYArray[sss]);
    }
   */
   //printf("=========================================\n");
   //printf("=========================================\n");
   //printf("=========================================\n");


       srand(time(0));
       //test("confusion",test_confusion_once);
       //test("diffusion",test_diffusion_once);
       //return
       int xxxxx = test_test_vector();



     //printf ("\nA============================ 1 \n");

//http://stackoverflow.com/questions/2614249/dynamic-memory-for-2d-char-
   array
//printf ("\n++++++++++++++++++ArraySizeInBytes %d\n", ArraySizeInBytes);
  MPI_Datatype BlockSize32;
  //MPI_Datatype SubArrOfBlockSize16;
  MPI_Status status;
  MPI_Init (&argc, &argv);
  MPI_Comm_rank (MPI_COMM_WORLD, &rank);
  MPI_Comm_size (MPI_COMM_WORLD, &size);
  int Cores = size - 1;      //TODO  deduct the one again
/*  if (Cores < 1)
   {
     Cores = 3;
   }*/
```

```
    //printf ("\nA============================ 0 \n");
  int V = 0;
  if (Cores > 0)
    V = ArraySizeInBytes / Cores;
       unsigned int Total_Mi= Total_M;
        //unsigned int ttmm= (unsigned int ) Total_M /Cores;
  unsigned int recordQuota = Total_Mi /Cores;
   Total_Mi = Cores * recordQuota;
  //unsigned int recordQuota = (unsigned int) recordQuotaX;
  //int MrecordPerCoreRem = ((int)ttmm % Cores); //Total_M % Cores;
       printf ("\n>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
          Cores %d ... Total_M %d .. recordQuota %d \n", Cores, Total_Mi,
          recordQuota);
  //if (MrecordPerCoreRem) {Total_M = Cores * recordQuota ;}
  //assert (MrecordPerCoreRem == 0);
//     printf ("\n>>>>>>>>>>>>>> Total_M %d .. MrecordPerCoreRem %d \n",
   Total_M, MrecordPerCoreRem);
//https://ubuntuforums.org/showthread.php?t=1430052
//http://www.sanfoundry.com/c-program-print-program-name-and-arguments/
  if (rank == 0)
    printf ("\nGREPME::: rank:%d Cores:%d usableCores:%d
       requiredMemoryInMegaBytes:%G memoryPerCoreInMegaBytes:%G PID:%d Ref
        %s \n",
          rank, size, Cores, getpid (), requiredMemoryInMegaBytes,
             memoryPerCoreInMegaBytes, argv[0]);

  int winner = 0;
  /* TODO Do I really need a different key, or should i send the initial
     key??? since I recreate the whole schedule i should probably send the
      initial key.
   *  *      TODO why do i need to flip the 2 n words at the end of
      decryption? */
  uint16_t text32[] = {
    0x6877,
    0x6565,
  };
  uint16_t CT32[] = {
    0x0001,
```

```
    0x0001,
  };
  uint16_t PT32_Dec[] = {
    0x0001,
    0x0001,
  };
  const uint16_t key64[] = {
    0x0100,
    0x0908,
    0x1110,
    0x1918,
  };
  const uint16_t key64_Dec[] = {
    0x7fbe,
    0xdb79,
    0x6a47,
    0xe5bf,
  };
  const uint16_t key32[] = {
    0x0100,
    0x0908,
  };
  const uint16_t key32_Dec[] = {
    0x7fbe,
    0xdb79,
  };
  const uint16_t key32_32_Dec[] = {
    0x8f0c,
    0xa487,
  };

  uint16_t lfsr48[]={ 0x0123, 0x4567, 0x89ab };
  uint16_t bit48[]={ 0x0000, 0x0000, 0x0000 };

  uint16_t start_state = 0xACE1u;   /* Any nonzero start state will work.
     */
//  uint32_t start_state32 = 0x434B434Bu;
```

```
  uint64_t start_state64 = 0x434B434B434B434Bu;
//uint64_t start_state64 = 0x0000555555FF0000u;

uint16_t lfsr = start_state;
uint32_t lfsr32 = start_state32;
uint64_t lfsr64 = start_state64;

uint16_t bit;              /* Must be 16bit to allow bit<<15 later in
   the code */
uint32_t bit32;
uint64_t bit64;            /* Must be 16bit to allow bit<<15 later in
   the code */
unsigned period = 0;
unsigned char buff[BlockSize + 1];
unsigned long long BruteForceNumberOfComputations = pow (2,
   BlockSizeBits); //Exhaustive search
unsigned long long WallTime = 48 * 60 * 60; // 2 days in unit-seconds
//double recordQuota = (double) ((double)ttmm / (double)Cores);
//int M = Total_M / Cores;
unsigned int M = recordQuota;
src = 0;
dest = 1;
tag = 0;
int tag_M2C_SP = 0;
int tag_M2C_LFSRSP = 3;
int tag_M2C_EP = 1;
int tag_C2M_WNR = 2;
unsigned char *TestKy;
unsigned char *TestPT;
unsigned char *TestCT;
unsigned char *TestRT;
TestKy = malloc (BlockSizeBytes);
TestPT = malloc (BlockSizeBytes);
TestCT = malloc (BlockSizeBytes);
TestRT = malloc (BlockSizeBytes);
```

```
  unsigned char **LFSRSP;
  unsigned char **SP;
  unsigned char **EP;
/*  int numbreOfChallenges =100;
  char **CTArray;
  char **KYArray;
  char PTArray[]= "Prof. Guang Gong, Ph.D., IEEE Fellow Department of
     Electrical and Computer Engineering Office: EIT 4158 Tel: (519)
     888-4567 x 35650 Fax: (519) 746-3077 Email: Homepage: http://comsec.
     uwaterloo.ca/~ggong/ (or access from ECE Dept. homepage) Prof. Mark
     Aagaard, Ph.D. Department of Electrical and Computer Engineering
     Office: DC 2539 Tel: (519) 888-4567 x 33138 Email: Homepage: https://
     ece.uwaterloo.ca/~maagaard/ (or access from ECE Dept. homepage) Our
     Research Areas Lightweight cryptography, cryptography and elliptic
     curve cryptography Signal design for wireless CDMA, OFDM and MIMO
     communications Security in cloud, network, ad-hoc network and RFID
     systems Wireless and multimedia communication security Security and
     privacy in machine-to-machine communication";
  KYArray = (char **) calloc ( numbreOfChallenges, sizeof (char *));
  if (KYArray == NULL)
    {
      fprintf (stderr, "not enough memory for Challenge Array\n");
    }
  char *blobzz = malloc ( numbreOfChallenges * BlockSizeBytes * sizeof (
     char));
  if (blobzz == NULL)
    {
      fprintf (stderr, "not enough memory for Challenge Array Blob\n");
    }
  else
    {
       memcpy (blobzz, PTArray , BlockSizeBytes * numbreOfChallenges );

    }
  int iizz = 0;
  for (iizz = 0; iizz < numbreOfChallenges; iizz++)
    {
```

```
     KYArray[iizz] = &blobzz[iizz * BlockSizeBytes];
    }
***********/


/*
 *  *http://stackoverflow.com/questions/25628321/dynamic-memory-allocation-
    in-mpi
 *   *http://stackoverflow.com/questions/5901476/sending-and-receiving-2d-
    array-over-mpi
 *   */
/*  CTArray = (char **) calloc ( numbreOfChallenges, sizeof (char *));
  if (CTArray == NULL)
    {
      fprintf (stderr, "not enough memory for Challenge Array\n");
    }
  char *blobzzz = malloc ( numbreOfChallenges * BlockSizeBytes * sizeof (
     char));
  if (blobzzz == NULL)
    {
      fprintf (stderr, "not enough memory for Challenge Array Blob\n");
    }
  else{

    }
  int iizzz = 0;
  for (iizzz = 0; iizzz < numbreOfChallenges; iizzz++)
    {
      char * Ress = (char *) malloc (BlockSizeBytes);
      CTArray[iizzz] = &blobzzz[iizzz * BlockSizeBytes];
      memcpy ( &CTArray[iizzz], Encrypt ( KYArray[iizzz], clear,
         BlockSizeBytes, Ress), BlockSizeBytes);
      free (Res);
    }
************/
/************/

  SP = (unsigned char **) calloc (recordQuota, sizeof (unsigned char *));
```

```
  if (SP == NULL)
    {
      fprintf (stderr, "not enough memory for SP\n");
    }
  unsigned char *blob = malloc (recordQuota * BlockSizeBytes * sizeof (
     unsigned char));
  if (blob == NULL)
    {
      fprintf (stderr, "not enough memory for SP Blob\n");
    }
  int ii = 0;
  for (ii = 0; ii < recordQuota; ii++)
    {
      SP[ii] = &blob[ii * BlockSizeBytes];
    }
//    printf("\n.%d.\n",ii);
//printf ("\nbfore2 ++++++++++++++++++++++++ArraySizeInBytes %d\n",
   ArraySizeInBytes);
  EP = (unsigned char **) calloc (recordQuota, sizeof (unsigned char *));
  if (EP == NULL)
    {
      fprintf (stderr, "not enough memory for EP\n");
    }
  unsigned char *blobEP = malloc (recordQuota * BlockSizeBytes * sizeof (
     unsigned char));
  if (blobEP == NULL)
    {
      fprintf (stderr, "not enough memory for EP Blob\n");
    }
  int iii = 0;
  for (iii = 0; iii < recordQuota; iii++)
    {
      EP[iii] = &blobEP[iii * BlockSizeBytes];
    }
//    printf("\n.%d.\n",iii);
//printf ("\nafter ++++++++++++++++++ArraySizeInBytes %d\n",
   ArraySizeInBytes);
  LFSRSP = (unsigned char **) calloc (Cores, sizeof (unsigned char *));
```

```
  if (LFSRSP == NULL)
    {
      fprintf (stderr, "not enough memory dor LFSRSP....really...\n");
    }
  unsigned char *blobLFSRSP = malloc (Cores * BlockSizeBytes * sizeof (
     unsigned char));
  if (blobLFSRSP == NULL)
    {
      fprintf (stderr, "not enough memory for LFSRSP Blob\n");
    }
  int iiii = 0;
  for (iiii = 0; iiii < Cores; iiii++)
    {
      LFSRSP[iiii] = &blobLFSRSP[iiii * BlockSizeBytes];
    }
/***********************************************/

  //printf    ("\nthread %d [a]++++++++++++++++++++++++++++++++\n", rank);
  MPI_Type_vector (BlockSizeBytes, 1, 1, MPI_UNSIGNED_CHAR, &BlockSize32);
       int xyza= sizeof(BlockSize32);
  ///MPI_Type_vector (1, BlockSizeBytes, 1, MPI_UNSIGNED_CHAR, &
     BlockSize32);
  printf    ("\nthread %d [b]++++++++++++++++++ sizeof BlockSize32 %d \n
     ", rank, xyza );
  MPI_Type_commit (&BlockSize32);
  //printf    ("\nthread %d [c
     ]++++++++++++++++++++++++++++++++++++++++++++++++++++++++\n", rank);


  //MPI_Type_vector (V, 1, 1, MPI_UNSIGNED_CHAR, &SubArrOfBlockSize16);
  //printf    ("\nthread %d [d
     ]++++++++++++++++++++++++++++++++++++++++++++++++++++++++\n", rank);
  //MPI_Type_commit (&SubArrOfBlockSize16);

  printf    ("\nthread %d [e]
     xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx\n", rank);

/** Master code */
```

```
  /*http://www.linux-mag.com/id/1332/ */
  if (!rank)
    {


/*
 *http://stackoverflow.com/questions/25628321/dynamic-memory-allocation-in
    -mpi
 *http://stackoverflow.com/questions/5901476/sending-and-receiving-2d-
    array-over-mpi
*/

     // printf ("\n4- Vector Chunk is %d as full SP array size in bytes is
         %d and usable cores are: %d\n",V, ArraySizeInBytes,Cores);

     /*http://stackoverflow.com/questions/1088622/how-do-i-create-an-array
        -of-strings-in-c */
     unsigned char *CipherTypes[2];
     CipherTypes[0] = "XOR___";
     CipherTypes[1] = "DES___";
     CipherTypes[2] = "SIMECK";
     CipherTypes[3] = "SPECK_";
     CipherTypes[4] = "SIMON_";
     CipherTypes[5] = "SHA3__";

     //printf("Types defined \n");
     unsigned char *EncProtocol;
     EncProtocol = malloc (7);
     memcpy (EncProtocol, CipherTypes[CipherUnderAttack], 6);
     //printf("String type defined \n");

     int y = 0;
     int sample = 10;
     double sumsum = 0;
     /*
        for (y=0; y<sample; y++){
        //TODO: fix testEncDecaverage time calculation
```

```
        sumsum=testEncDec( rank, BlockSizeBytes, TestKy, TestCT, TestPT ,
           TestRT);
        }
        printf("\n========================= Tested %d %s encryption %d
           times and the average is %d \n",CipherUnderAttack, EncProtocol,
            sample, (sumsum/sample));
      */
     //printf ("\n6- Vector Chunk is %d as full SP array size in bytes is
        %d and usable cores are: %d\n",V, ArraySizeInBytes,Cores);

     printf ("========================= /** Initializing the arrays*/ \n
        ");
     /*http://stackoverflow.com/questions/2279379/how-to-convert-integer-
        to-char-in-c */
     /* for (i = 0; i < ArraySizeInBytes; i++){
        SP[i] = i + '0';
        EP[i] = i + '0';
        } */
/** Visual verification*/
     /*for (i = 0; i < ArraySizeInBytes; i=i+BlockSizeBytes){ //printf(" %
        c", SP[i]);
        printf ("[C%d/%d][M%d/%d/%d][T%d/%d/%d/%d]\t[LFSR][lfsr: %llu, \t
           %s \tSP=%c%c=\tEP=%c%c=]\n", rank, Cores, i, M, Total_M, 0, T,
           WallTime, BruteForceNumberOfComputations, lfsr, binrep( lfsr ,
           buff , BlockSize ),SP[i+0],SP[i+1],EP[i+0],EP[i+1]);
        } */
     printf
       ("========================= /** Done Initializing the arrays*/ \n
          ");

     before_lfsr_seed = clock ();
//printf("10- BlockSizeBytes: %d, period: %d , V= %d\n",BlockSizeBytes,
   period, V);

     printf
       ("=========================/** Randomizing the array values usinf
          LFSRs for specific Block Sizes*/ \n");
/** Randomizing the array values usinf LFSRs for specific Block Sizes*/
```

```
/*http://stackoverflow.com/questions/16891655/galois-lfsr-explanation
   -of-code
   https://en.wikipedia.org/wiki/Linear-feedback_shift_register */
if (BlockSize == 16)
  {
    period = 0;
    do
      {
        //printf("Inside B=16 LFSR\n");
        /* taps: 16 14 13 11; feedback polynomial: x^16 + x^14 + x^13
           + x^11 + 1 */
        /*      bit  = ((lfsr >> (16-16)) ^ (lfsr >> (16-14)) ^ (lfsr
           >> (16-13)) ^ (lfsr >> (16-11)) ) & 1; */
        bit =
          ((lfsr >> 0) ^ (lfsr >> 2) ^ (lfsr >> 3) ^ (lfsr >> 5)) & 1;
        lfsr = (lfsr >> 1) | (bit << (BlockSize - 1));
        /*http://stackoverflow.com/questions/13279024/convert-a-uint16
           -t-to-char2-to-be-sent-over-socket-unix */
        unsigned char lo = lfsr & 0xFF;
        unsigned char hi = lfsr >> 8;
        memcpy (&SP[period + 0], &lo, 1);
        memcpy (&SP[period + 1], &hi, 1);
        //      printf ("[C%d/%d][M%d/%d/%d][T%d/%d/%d/%d]\t[LFSR][
           lfsr: %llu, \t %s \tSP=%c%c=\tEP=%c%c=]\n", rank, Cores,
           period, M, Total_M, 0, T, WallTime,
           BruteForceNumberOfComputations, lfsr, binrep( lfsr , buff
           , BlockSize ),SP[period+0],SP[period+1],EP[period+0],EP[
           period+1]);
        period = period + BlockSizeBytes;
        // while ( (lfsr != start_state) && (period <ArraySizeInBytes)
           );
      }
    while (period < ArraySizeInBytes);
  }
else if (BlockSize == 8)
  {
  }
else if (BlockSize == 48)
```

```
{
  period = 0;
  int lfsrspcount = 0;
  do
    {
      /*char lo = lfsr48 & 0xFF;
      char b1 = lfsr48 >> 8;
      char b2 = lfsr48 >> 16;
      char b3 = lfsr48 >> 24;
      char b4 = lfsr48 >> 32;
      char hi = lfsr48 >> 40;*/
      int rq = ( int)recordQuota;
      int  remainder = period % rq; //(Total_M / Cores);
      if (remainder == 0)
        {
          memcpy (LFSRSP[lfsrspcount], &lfsr64, BlockSizeBytes);
          /*printf
            ("\n________Rank %d period %d ------>>> Copied lfsr: %4x
                into LFSRSP[%d] \t",
             rank, period, lfsr64, lfsrspcount);
           */
          unsigned char *SPaddrs = LFSRSP[lfsrspcount];
          int blks = 0;
          for (blks = 0; blks < BlockSizeBytes; blks++)
            {
              printf ("____%c", *SPaddrs);
              SPaddrs++;
            }
          printf ("\n");
          lfsrspcount++;
        }
      else
        {}
      period = period + 1;
      bit64 =
        (( lfsr64 >> (BlockSizeBits - 32)) ^
         ( lfsr64 >> (BlockSizeBits - 7)) ^
         ( lfsr64 >> (BlockSizeBits - 3)) ^
```

```
          ( lfsr64 >> (BlockSizeBits - 2))) & 1;
        lfsr64 = ( lfsr64 >> 1) | (bit64 << (BlockSizeBits - 1));
      }
    while (period < Total_Mi);  //ArraySizeInBytes);
    printf ("M======= we just LFSRd %d times. >>>>>\n", period);
  }
else if (BlockSize == 32)
  {
    period = 0;
    int lfsrspcount = 0;
    /* https://en.wikipedia.org/wiki/Linear-feedback_shift_register
       */
    do
      {
        unsigned char lo = lfsr32 & 0xFF;
        unsigned char b1 = lfsr32 >> 8;
        unsigned char b2 = lfsr32 >> 16;
        unsigned char hi = lfsr32 >> 24;
        int remainder = period % (Total_M / Cores);
        if (remainder == 0)
          {
            memcpy (LFSRSP[lfsrspcount], &lfsr32, BlockSizeBytes);

            printf
              ("\nRank %d period %d ------>>> Copied lfsr: %4x into
                 LFSRSP[%d] \t",
               rank, period, lfsr32, lfsrspcount);
            unsigned char *SPaddrs = LFSRSP[lfsrspcount];
            int blks = 0;
            for (blks = 0; blks < BlockSizeBytes; blks++)
              {
                printf ("_%c", *SPaddrs);
                SPaddrs++;
              }
            printf ("\n");

            lfsrspcount++;
          }
```

```
            else
              {
                //       printf ("\nRank %d period %d ------>>> skip lfsr:
                     %4x \n", rank, period, lfsr32 );
              }
            /* printf("period_%d_address_%d_valueiShouldBe_%04x_%c_%c_%c_%
               c____AndItIs___",period,SP[period],lfsr32,lo,b1,b2,hi);
               char *adrs= SP[period];
               /int b=0;
               for (b=0; b<BlockSizeBytes; b++){ printf("_%c",*adrs); adrs
                  ++;}
               printf("\n");
             */
            period = period + 1;    //BlockSizeBytes;

            /*primitive polynomial from Kali: y^32 + y^15 + y^9 +       y
               ^7 + y^4 +                  y^3 +
                                             1 */
            //tap sequence [32, 7, 3, 2, 0]
            /* wikipedia polynomial=      x^32   +                    x
               ^7                      +   X^3                    +    X
               ^2  + X^0 */
            bit32 =
              ((lfsr32 >> (BlockSizeBits - 32)) ^
               (lfsr32 >> (BlockSizeBits - 7)) ^ (lfsr32 >>
                                                  (BlockSizeBits -
                                                   3)) ^ (lfsr32 >>
                                                          (BlockSizeBits -
                                                           2))) & 1;
            lfsr32 = (lfsr32 >> 1) | (bit32 << (BlockSizeBits - 1));
/////////XXXX printf (" %d",period);
           }
         while (period < Total_M);  //ArraySizeInBytes);
         printf ("M======= we just LFSRd %d times. >>>>>\n", period);
       }
     else if (BlockSize == 40)
       {
       }
```

```
else if (BlockSize == 48)
  {
    /*Kali primitive polynomial: y^48 + y^25 + y^23 + y^17 + y^12 + y
       ^11 + y^10 + y^8 + y^7 + y^3 + 1 */
    /* y^96 + y^63 + y^61 + y^60 + y^59 + y^54 + y^50 + y^46 + y^44 +
        y^40 + y^37 + y^33 + y^32 + y^28 + y^24 + y^23 + y^21 + y^20
       + y^19 + y^18 + y^16 + y^15 + y^14 + y^13 + y^11 + y^9 + y^8 +
        y^7 + y^6 + y^4 + y^3 + y^2 + 1
     *
     * y^128 + y^7 + y^2 + y + 1 */
  }
else if (BlockSize == 56)
  {
  }
else if (BlockSize == 64)
  {
    period = 0;
    do
      {
        /* polynomial=  x^56   +   x^2         +  X^4          +
              X^7           + 1  */
        bit64 =
          ((lfsr64 >> (BlockSizeBits - 56)) ^
           (lfsr64 >> (BlockSizeBits - 7)) ^ (lfsr64 >>
                                              (BlockSizeBits -
                                               4)) ^ (lfsr64 >>
                                                      (BlockSizeBits -
                                                       2))) & 1;
        lfsr64 = (lfsr64 >> 1) | (bit64 << (BlockSizeBits - 1));
        //printf ("\nlfsr: %llu, %s", lfsr64, binrep( lfsr64 , buff ,
           BlockSizeBits ));
        unsigned char lo = lfsr64 & 0xFF;
        unsigned char b1 = lfsr64 >> 8;
        unsigned char b2 = lfsr64 >> 16;
        unsigned char b3 = lfsr64 >> 24;
        unsigned char b4 = lfsr64 >> 32;
        unsigned char b5 = lfsr64 >> 40;
        unsigned char b6 = lfsr64 >> 48;
```

```
            unsigned char hi = lfsr64 >> 56;

            memcpy (&SP[period + 0], &lo, 1);
            memcpy (&SP[period + 1], &b1, 1);
            memcpy (&SP[period + 2], &b2, 1);
            memcpy (&SP[period + 3], &b3, 1);
            memcpy (&SP[period + 4], &b4, 1);
            memcpy (&SP[period + 5], &b5, 1);
            memcpy (&SP[period + 6], &b6, 1);
            memcpy (&SP[period + 7], &hi, 1);
            period = period + BlockSizeBytes;

          }
        while (period < ArraySizeInBytes); //( (lfsr64 != start_state64)
           && (period <Total_M));
      }
    printf
      ("\nP=========================/** Done Randomizing the array
         values usinf LFSR*/ \n");
/*
      int p=0; for (p=0; p< Total_M; p++){
             char *adrs= SP[p];
             int bb=0;
             for (bb=0; bb<BlockSizeBytes; bb++){ printf("_%c",*adrs);
                adrs++;}
             printf("\n");
      }
*/
    before_dispatch = clock ();
    master_time_to_seed =
      ((float) (before_dispatch - before_lfsr_seed)) / CLOCKS_PER_SEC;
    printf ("\n\tmaster_time_to_seed \t : %G \n\n\n ",
           master_time_to_seed);

    printf
      ("\nP=========================/** Define the Dispatch vector**/ of
          size V= %d\n",
       V);
```

```
/** Define the Dispatch vector**/
/** Dispatch to child processes */
     int r = 0;
     for (r = 1; r <= Cores; r++)
       {
         int offset = (r - 1) * recordQuota;
         int chunkkk = recordQuota * sizeof (unsigned char *);
         unsigned char * adrss= CTchallenge; //LFSRSP[r - 1]; //SP[offset
            ];
            /*printf("FINDME ....========== master Process sending Vector
                offset %d and first pointer SP is:%d and value is \t",
               offset, adrss);
            int bbb=0;
            for (bbb=0; bbb<BlockSizeBytes; bbb++){ printf("__%d_%02x_ ",
               bbb, *adrss); adrss++;}
            printf("\n");
            */
/////XXXXX
         printf("master will send to slave\t\t\t ");
         MPI_Send (LFSRSP[r - 1], 1, BlockSize32, r, tag_M2C_LFSRSP,
            MPI_COMM_WORLD);
         printf("master finished sending.....\n ");
         //MPI_Send(SP[offset], 1 , SubArrOfBlockSize16, r, tag_M2C_SP,
            MPI_COMM_WORLD);
         //MPI_Send(EP[offset], 1 , SubArrOfBlockSize16, r, tag_M2C_EP,
            MPI_COMM_WORLD);

         //printf("Sent Core %d SP and EP too \n", r);
       }
     printf
       ("\n==========================================\nP
          ========================/** Done dispatching and creating file
          **/ \n");

     //MPI_Type_free(&Data16Type);
     /* http://www.cplusplus.com/reference/cstdio/fopen/ */
     FILE *pFile;
     pFile = fopen ("/scratch/m/maagaard/knassar/winners_32bit.txt", "w");
```

```
printf
  ("\nP=========================/**Created results file and Waiting
     for Child processes**/ \n");


before_results = clock ();
winner = 0;
for (r = 1; r <= Cores; r++)
  {
    MPI_Recv (&winner, 1, MPI_INT, r, tag_C2M_WNR, MPI_COMM_WORLD,
              &status);
    //printf("\nP========================= /** Received Process %d
       results %d*/ ", r, winner)

    /* if ( ((r-1)*recordQuota) == winner ) {
       printf("\t[[]]Yaaay winner %d \n", winner);
       if (pFile!=NULL){
       char    logEntry[100];
       sprintf( logEntry, "False positive Winner: [Thread %d ][win %d
          ]-\n", r, winner);
       fputs ( logEntry, pFile);
       }else{
       printf("Could not write to file..\n");
       }


       } else */
    if (winner != -1)
      {
        printf ("\t[[]]Yaaay winner %d \n", winner);
        if (pFile != NULL)
          {
            char logEntry[100];
            sprintf (logEntry, "Winner: [Thread %d ][win %d]-\n", r,
                     winner);
            fputs (logEntry, pFile);
            //     fputs ("winner is : ",pFile);
            //                   }else{
```

```
                //
              }
            else
              {
                printf ("Could not write to file..\n");
              }
          }
      }
    printf
      ("\nP==== Collected results, closing the result file and shutting
         down\n");
    fclose (pFile);
    //MPI_Abort( MPI_COMM_WORLD , 0);
    MPI_Finalize ();
    return 0;

  }
 else
  {                          /*child process */
    awaiting_dispatch = clock ();
    int offset = (rank - 1) * recordQuota;
    int chunkk = recordQuota * sizeof (unsigned char *);
    printf   ("C silly ========================== /** Child Process %d
       waiting */ \t\t and CT is: ", rank);
    unsigned char * verifyCTCptr = CTchallenge ; int sillycounter=0;
    for (sillycounter=0; sillycounter<BlockSizeBytes; sillycounter++){
       printf("%d_%02x ",sillycounter, *verifyCTCptr); verifyCTCptr++;}
      printf("\n");


    //sleep (rank);
    //MPI_Recv (LFSRSP[rank - 1], 1, BlockSize32, src, tag_M2C_LFSRSP,
               MPI_COMM_WORLD, &status);
    MPI_Recv (LFSRSP[rank - 1], 1, BlockSize32, src, tag_M2C_LFSRSP,
       MPI_COMM_WORLD, &status);
    printf   ("C silly ========================== rank %d done waiting
       <<<<<<<<<< \n ", rank);
/////XXXXX
```

```
//   MPI_Recv(SP[offset], 1 , SubArrOfBlockSize16, src, tag_M2C_SP,
   MPI_COMM_WORLD, &status);
//   MPI_Recv(EP[offset], 1 , SubArrOfBlockSize16, src, tag_M2C_EP,
   MPI_COMM_WORLD, &status);

{
     unsigned char *Resss = (unsigned char *) malloc (BlockSizeBytes);
     unsigned char *CTchallengess = (unsigned char *) malloc (
        BlockSizeBytes);
  memcpy (CTchallengess, Encrypt (ChallengeKey, clear, BlockSizeBytes,
     Resss),
         BlockSizeBytes);
     unsigned char * verifyCTCptrxss = CTchallengess ; int rxss=0;

     printf ("C silly Afterr ====== CT forChallenge (calc inside child) is
        : ");
     //for (rxss=0; rxss<BlockSizeBytes; rxss++){ printf("%d_%02x ",rxss,
        *verifyCTCptrxss); verifyCTCptrxss++;}
     printf ("\n");

       free (Resss);
     free (CTchallengess);
}









     before_randomizing = clock ();
//     printf
//     ("C========================= Child Process %d received and will
   calculate lfsr \n",
//      rank);
```

```
     unsigned char * adrss= LFSRSP[rank - 1]; //SP[offset];
        int bbb=0;
        /*printf("FINDME....====================== /** Child Process %d
           received Vector offset %d and first pointer SP is:%d and value
           is ", rank, offset, adrss);
        for (bbb=0; bbb<BlockSizeBytes; bbb++){ printf("%d_%c ",bbb, *
           adrss); adrss++;}
        printf("\n");
        */
     printf
       ("C========================== Child Process %d received and will
          calculate lfsr \n",
        rank);

/*
      adrss= SP[0];
        printf("FINDME....====================== /** Child Process %d
           first lfsr value in SP is: ", rank);
        bbb=0;
        for (bbb=0; bbb<BlockSizeBytes; bbb++){ printf("%d_%c ",bbb, *
           adrss); adrss++;}
        printf("\n");
*/
     calculateLFSR (BlockSizeBits, LFSRSP, rank, SP, recordQuota, lfsr32);


      adrss= SP[0];
        /*printf("FINDME....====================== /** Child Process %d
           first lfsr value in SP is: ", rank);
        bbb=0;
        for (bbb=0; bbb<BlockSizeBytes; bbb++){ printf("%d_%c ",bbb, *
           adrss); adrss++;}
        printf("\n");
        */

     before_calculating = clock ();
     printf
```

```
       ("C========================== /** Start calculating EP at Child
          Process %d for T= %d*/ \n",
        rank, T);

///TODO LOOP 3AK
     for (i = 0; i < recordQuota; i = i + 1)
       {
         int GlobalRecord = ((rank - 1) * recordQuota) + i;
         if ((i % 100000) == 0)
           printf ("C=========================Calculating at core %d
              record %d which is %d\n", rank, i, GlobalRecord); //TODO
              comeback to this statement oneday. disturbance in the force.
         unsigned char *address = SP[i];
         unsigned char *addressEP = EP[i];

/*
       printf ("\nC...before calc ...... record %d SPp addr %d , value ",i
          , address );
              int bbbb=0;
              for (bbbb=0; bbbb<BlockSizeBytes; bbbb++){ printf("_%c",*
                 address); address++;}
       printf ("...... EPp addr %d , value ", addressEP );
               for (bbbb=0; bbbb<BlockSizeBytes; bbbb++){ printf("_%c",*
                  addressEP); addressEP++;}

              printf("\n");
*/
         int result =
           CalculateEndPoint (SP[i], EP[i], BlockSizeBytes, i, rank,
                              BruteForceNumberOfComputations, M, WallTime,
                              T);

      address = SP[i];
      addressEP = EP[i];
       if (debug == 2) {
              printf ("\nC Rank %d : after calculating endpoints ........
                 record %d SPp addr %d , value ", rank, i, address );
              int bbbb=0;
```

```
              //for (bbbb=0; bbbb<BlockSizeBytes; bbbb++){ printf("___[%c_
                 %02x]",*address,*address); address++;}
              for (bbbb=0; bbbb<BlockSizeBytes; bbbb++){ printf("___[%02x
                 ]",*address); address++;}
              printf ("..... EPp addr %d , value ", addressEP );
              //for (bbbb=0; bbbb<BlockSizeBytes; bbbb++){ printf("___[%c_
                 %02x]",*addressEP ,*addressEP); addressEP++;}
              for (bbbb=0; bbbb<BlockSizeBytes; bbbb++){ printf("___[%02x
                 ]",*addressEP); addressEP++;}
              printf("\n\n");
              }



      }

/*   printf("C========================== Visual confirmation for
   calculated SP/EP %d \n", rank);
   for (i = offset; i < (offset+recordQuota); i=i+1){
       char * SPaddr = SP[i];
       char * EPaddr = EP[i];
       int blk=0;
       for (blk=0; blk<BlockSizeBytes; blk++){ printf("_%c",*SPaddr);
          SPaddr++;}
       for (blk=0; blk<BlockSizeBytes; blk++){ printf("_%c",*EPaddr);
          EPaddr++;}

   }
*/
     before_sorting = clock ();
     if (skipSort == 0)
       {
         printf
           ("C========================== /** Start Sorting EPs at Child
              Process %d */ \n",
            rank);
         /*for(i=offset; i<(offset+V); i=i+BlockSizeBytes) {
```

```
   printf("Mixxed: [Thread %d ][offset %d][chunksize %d][Record %
      d]----[SP:: %c%c ][ EP: %c%c ]\n", rank, offset, V, i, SP[i
      +0], SP[i+1], EP[i+0], EP[i+1]);
   } */
/*http://stackoverflow.com/questions/12646734/how-to-sort-an-
   array-of-string-alphabetically-case-sensitive-nonstandard-
   colla */
//printf("\nBefore Sorting
   .....................................................\n");
int j = 0;
unsigned char *epi;
epi = malloc (BlockSizeBytes + 1); // the extra 1 is for the null
unsigned char *epj;
epj = malloc (BlockSizeBytes + 1); // the extra 1 is for the null
int ret = strcmp (epi, epj);

for (i = 0; i < recordQuota; i = i + 1)
  {
    if ((i % 10000) == 0)
      printf ("C============== sorting at core %d record %d\n",
              rank, i);
    for (j = i + 1; j < recordQuota; j = j + 1)
      {
        //epi = malloc(BlockSizeBytes+1); // the extra 1 is for
           the null
        memcpy (epi, EP[i], BlockSizeBytes);
        epi[BlockSizeBytes] = '\0';
        //char * epj;
        //epj = malloc(BlockSizeBytes+1); // the extra 1 is for
           the null
        memcpy (epj, EP[j], BlockSizeBytes);
        epj[BlockSizeBytes] = '\0';
        int ret = strcmp (epi, epj);
        //       int ret = strcmp(&EP[i], &EP[j]);
        //TODO: when do those string finish? do i need to copy
           into another string with \0 at the end?
        //what happens if the sort parially works? binary search
           wouldnt be good. currently I think the search is
```

```
                   sequential so it doesn't matter.
                if (ret < 0)
                  {
                    //https://www.tutorialspoint.com/c_standard_library/
                       c_function_strcmp.htm
                    //      printf("str1 is less than str2\n");
                  }
                else if (ret > 0)
                  {
                    //  printf("str2 is less than str1\n");
                    unsigned char *tmp;
                    tmp = malloc (BlockSizeBytes);
                    memcpy (tmp, EP[i], BlockSizeBytes);
                    memcpy (EP[i], EP[j], BlockSizeBytes);
                    memcpy (EP[j], tmp, BlockSizeBytes);
                    memcpy (tmp, SP[i], BlockSizeBytes);
                    memcpy (SP[i], SP[j], BlockSizeBytes);
                    memcpy (SP[j], tmp, BlockSizeBytes);
                    free (tmp);
                  }
                else
                  {
                    //      printf("str1 is equal to str2\n");
                  }
              }
          }
        free (epi);
        free (epj);
        /*for(i=offset; i<(offset+V); i=i+BlockSizeBytes) {
           printf("Sorted: [Thread %d ][offset %d][chunksize %d][Record %
               d]----[SP:: %c%c ][ EP: %c%c ]\n", rank, offset, V, i, SP[i
               +0], SP[i+1], EP[i+0], EP[i+1]);
           } */
        if (debug >= 2) printf
          ("C========================= /** Visual confirmation for
             sorted SP/EP %d */ \n",
           rank);
/*
```

```
   for (i = 0; i < recordQuota; i=i+1){
       char * SPaddr = SP[i];
       char * EPaddr = EP[i];
       char * CTaddr = CTchallenge;
       int blk=0;
       printf ("\n>>>111_____");
       for (blk=0; blk<BlockSizeBytes; blk++){ printf("_%c",*SPaddr);
          SPaddr++;}
       printf ("________222________");
       for (blk=0; blk<BlockSizeBytes; blk++){ printf("_%c",*EPaddr);
          EPaddr++;}
       printf ("_________333______");
       for (blk=0; blk<BlockSizeBytes; blk++){ printf("_%c",*CTaddr);
          CTaddr++;}

       printf("\n");
   }

*/
        printf
          ("\n\nC========================== DONE Sorting at core %d
             <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<< \n",
           rank);
      }
    else
      {
        printf
          ("\n\nC=================No Sorting at core %d. We chose to save
              cycles. \n",
           rank);
      }
    before_searching = clock ();

    winner =
      searchEPforCT (SP[0], EP[0], rank, offset, V, ReducedCTC,
         CTchallenge,
                    T, BlockSizeBytes, ChallengeKey, CTchallenge);
    before_reporting = clock ();
```

```
     child_time_to_search =
       ((float) (before_reporting - just_started)) / CLOCKS_PER_SEC;
     printf
       ("\n\tChildTime %d to finish searching \t : %d just started %d and
          before reporting time is %d \n\n\n ",
        rank, master_time_to_seed, just_started, before_reporting);


    int sss=0;
    for (sss=0; sss<numbreOfChallenges; sss++){
      printf("SEARCH_ARRAY :: BEGIN ::i rank %d :: count:: %d <==> %c
         <==> %c \n",rank, sss,CTArray[sss], KYArray[sss]);
      int winner =
                searchEPforCT (SP[0], EP[0], rank, offset, V, (unsigned
                   char *)CTArray[sss], (unsigned char *)CTArray[sss],
                   T, BlockSizeBytes,(unsigned char *) KYArray[sss], (
                      unsigned char *)CTArray[sss]);
      printf("SEARCH_ARRAY :: END ::i rank %d :: count:: %d <==> %c <==>
         %c \n",rank ,sss,CTArray[sss], KYArray[sss]);

    }
/* KHAXAN
 *Khaled
              for loop all 100 challenges and call search for each
TODO
 * */

     if (winner > -1)
       {
         ////printf("\nC===>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>> Winner of
             %d is ... %d\n", rank, winner );
       }
     else
       {
         ////printf("\nC===>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>> No Winner
             of %d is ... %d\n", rank, winner );
       }
     MPI_Send (&winner, 1, MPI_INT, 0, tag_C2M_WNR, MPI_COMM_WORLD);
```

```
     //printf("\nAfter 3.....................................%d
        ...\n", rank);

     //MPI_Abort( MPI_COMM_WORLD , 0);
     MPI_Finalize ();
     return 0;
     //printf( "C==== I %d sent to Boss \n", rank);

     /* printf("\n
        ============================================================
        Child Process %d DONE and reported winner%d \n", rank, winner);
        */
   }
// DONE TODO cleanup the false alarm.. dealr with 0 return
// DONE TODO add other ciphers
// DONE TODO add multiple LFSR's for multiple block sizes
// DONE TODO allign, indent better
// TODO, use estimation code from tmt17 to use more CPU and memory
// TODO redo DES, get its lfsr fixed

  MPI_Finalize ();
  printf (" ...t.........X...........AFTER MPI finalize \n");
  return 0;
  just_finished = clock ();
}
scinet03-ib0-$
```

C.2 The Simeck program after doing some changes to allow decryption and smaller key scheduling key

```
gpc-f102n004-ib0-$ cat simeck48.h

#ifndef SIMECK48_H
#define SIMECK48_H
```

```
#include <stdint.h>

#define LROT24(x, r) ((((x) << (r)) % (1 << 24)) | ((x) >> (24 - (r))))

#define ROUND48(key, lft, rgt, tmp) do { \
    tmp = (lft); \
    lft = ((lft) & LROT24((lft), 5)) ^ LROT24((lft), 1) ^ (rgt) ^ (key); \
    rgt = (tmp); \
} while (0)

#define REV_ROUND48(key, lft, rgt, tmp) do { \
    tmp = (rgt); \
    rgt = ((rgt) & LROT24((rgt), 5)) ^ LROT24((rgt), 1) ^ (lft) ^ (key); \
    lft = (tmp); \
} while (0)

void simeck_48_96(      const uint32_t master_key[],     const uint32_t
   plaintext[],     uint32_t ciphertext[] ) {
    const int NUM_ROUNDS = 36;
    int idx;
    uint32_t keys[4] = {      master_key[0],      master_key[1],
       master_key[0],      master_key[1],  };
    ciphertext[0] = plaintext[0];
    ciphertext[1] = plaintext[1];
    uint32_t temp;
    uint32_t constant = 0xFFFFFC;
    uint32_t sequence = 0x9A42BB1F;
    for (idx = 0; idx < NUM_ROUNDS; idx++) {
        ROUND48(
               keys[0],
               ciphertext[1],
               ciphertext[0],
               temp
        );
        constant &= 0xFFFFFC;
        constant |= sequence & 1;
        sequence >>= 1;
        ROUND48(
```

```
                constant,
                keys[1],
                keys[0],
                temp
        );
        temp = keys[1];
        keys[1] = keys[2];
        keys[2] = keys[3];
        keys[3] = temp;
    }
}

void simeck_48_96_Dec(      const uint32_t master_key[],     const
   uint32_t plaintext[],     uint32_t ciphertext[] ) {
    const int NUM_ROUNDS = 36;
    int idx;
    uint32_t keys[4] = {      master_key[0],      master_key[1],
       master_key[0],      master_key[1],  };
    ciphertext[0] = plaintext[0];
    ciphertext[1] = plaintext[1];
    uint32_t temp;
    uint32_t constant = 0xFFFFFC;
    uint32_t sequence = 0x9A42BB1F;
    uint32_t KeySchedule[NUM_ROUNDS];
    for (idx = 0; idx < NUM_ROUNDS; idx++) {
        KeySchedule[35 - idx]= keys[0],
        constant &= 0xFFFFFC;
        constant |= sequence & 1;
        sequence >>= 1;
        ROUND48(
                constant,
                keys[1],
                keys[0],
                temp
        );
        temp = keys[1];
        keys[1] = keys[2];
        keys[2] = keys[3];
```

```
        keys[3] = temp;
    }
    for (idx = 0; idx < NUM_ROUNDS; idx++) {
        REV_ROUND48(
                KeySchedule[idx],
                ciphertext[1],
                ciphertext[0],
                temp
        );
    }

}

#endif // SIMECK48_H
```