

Towards Peer-to-Peer-based Cryptanalysis

Matthäus Wander, Arno Wacker, and Torben Weis

University of Duisburg-Essen, Distributed Systems Group,
Bismarckstraße 90, 47057 Duisburg, Germany
{matthaeus.wander|arno.wacker|torben.weis}@uni-due.de

Abstract—Modern cryptanalytic algorithms require a large amount of computational power. An approach to cope with this requirement is to distribute these algorithms among many computers and to perform the computation massively parallel. However, existing approaches for distributing cryptanalytic algorithms are based on a client/server or a grid architecture. In this paper we propose the usage of peer-to-peer (P2P) technology for distributed cryptanalytic calculations. Our contribution in this paper is three-fold: We first identify the challenges resulting from this approach and provide a classification of algorithms suited for P2P-based computation. Secondly, we discuss and classify some specific cryptanalytic algorithms and their suitability for such an approach. Finally we provide a new, fully decentralized approach for distributing such computationally intensive jobs. Our design takes special care about scalability and the possible untrustworthy nature of the participating peers.

Index Terms—Peer-to-peer, cryptanalysis, distributed computing.

I. INTRODUCTION AND GOALS

The goal of our project is to develop a set of tools which allows for running large cryptanalytic jobs on a peer-to-peer (P2P) system. While P2P systems are known to scale well (BitTorrent, Skype), they are much harder to deal with than server-based systems, based on grids or cloud computing offerings. The reason to build on top of P2P nevertheless is to circumvent the inherent cost of any server-based solution. We want to allow individuals and researchers to tap into the power of high performance computing for cryptanalysis without the need to own or rent computers for this purpose. Each peer can inject a cryptanalytic job and hope that other peers will contribute to this job by investing their CPU and bandwidth. In return, each peer must be willing to help with other peers' jobs from time to time. Mechanisms for achieving fairness and fighting freeriders is beyond the scope of this paper and this issue has been successfully addressed by many modern P2P systems, for example BitTorrent.

The stakeholder in our approach is a user with cryptanalytic interest and experience, but with little knowledge of large scale distributed computing. Here P2P systems become helpful because they are known to be self-organizing, i.e. they can fulfill some service without any centralized administrator. Furthermore, they can handle drop outs and failures of many peers. While these events can temporarily slow down a P2P system, it will eventually and automatically recover from these failures which makes it self-stabilizing and therefore very useful in our scenario.

Thus, we can formulate two key requirements: 1) Running a large-scale distributed cryptanalytic algorithm must be simple enough for our stakeholders, i.e. it must not require expert knowledge in distributed computing. 2) Each individual should be able to inject cryptanalytic jobs in the system and support other users' jobs with local computing resources. As discussed in the related work section, these requirements are only fulfilled by a P2P system.

In this paper we show that P2P systems – while fulfilling our requirements – pose new challenges which do not exist in server-based solutions in this form. We iterate over different algorithm designs and discuss the implications of executing algorithms of these classes on a P2P system. Based on this analysis we discuss two cryptanalytic algorithms and their suitability for P2P-based computation. Additionally, we present a new fully decentralized approach for distributing the discussed algorithms in a P2P system. Our approach is specifically tailored towards scalability and different failure classes caused by malicious or unreliable peers.

Our paper is structured as follows: In the next section we discuss related approaches for distributed computing. After that, we present in Section III the emerging challenges when using P2P-based distribution instead of client/server- or grid-based approaches. With these challenges in mind, we discuss in Section IV different algorithm classes suitable for distributed computing. Furthermore, we analyze the suitability of specific cryptanalytic algorithms for P2P-based distributed computing. Based on this analysis we present in Section V our new fully decentralized approach for P2P-based distributed cryptanalysis. Finally, we conclude our work in Section VI with a brief summary and thoughts about future work.

II. RELATED WORK

In this section, we discuss other existing models for distributed computing and their suitability for the scenario explained above.

Client/server computing: In the conventional client/server model the client connects to a central server (or server cluster) which in return provides some service to the client. The distributed computing platform BOINC [1] is based on the client/server model, however extends it in two ways. 1) The client is not a pure consumer, but provides his computation service to the server. 2) The client may connect to multiple servers in parallel and offer his resources to them. The BOINC design perfectly fits its goal to utilize computing power donated by

the participants for long-term projects (*volunteer computing*). The servers run a service to accommodate and manage the donated resources. While every participant is able to provide his own server to distribute a computational job (meeting the second requirement), there is a considerable amount of administration effort to setup this server. This is not suitable in regard to our first requirement. A BOINC extension called nuBOINC [2] allows the user to submit new jobs with an arbitrary input to the (customized) BOINC server. The server distributes the user-submitted jobs to the participants, provided that the application to be run is installed on the participant's computers. This however requires that the user has access to a BOINC server providing this type of distribution service. We do not expect participants to setup a service for others if there is no incentive to do so.

The distributed.net project [3] is a platform for distributed computing of cryptanalytic tasks, among others. It is based on the client/server model and is restricted to applications distributed by the project operator. Participants can donate resources to the project but do not make use of donated resources.

Cloud computing: In recent times the concept of cloud computing attracted some attention [4]. The idea is to move the computing resources from the service provider to an infrastructure provider. The service provider may dynamically customize the resource capacities bought from the infrastructure provider subject to the service demand. This introduces an additional abstraction level compared to the client/server principle, however still requires a service provider to setup and administrate its service. Thus, cloud computing shows for our purpose similar characteristics as the client/server principle. The necessity to buy cloud resources from the infrastructure provider adds an additional drawback.

Cluster computing: The purpose of cluster computing is to aggregate available resources of multiple computers under control of one authority. The typical usage of cluster computing is to achieve high performance of a supercomputer with "off-the-shelf" computers. Due to the central administration of resources cluster computing does not fit our scenario.

Grid computing: The idea of grid computing is a "*co-ordinated resource sharing and problem solving in dynamic, multi-institutional virtual organizations*" [5]. Grid computing can be seen as a larger-scale cluster computing between different institutions. Unlike cluster computing grids comprise heterogeneous resources, for example single PCs, clusters or supercomputers. Similar to cluster computing grids require a manual setup, thus not being suitable to fulfill our first requirement.

Grid vs. P2P computing: Grids and P2P systems share some similarities like heterogeneous resources or the absence of a central administration. However grids and P2P systems emerged in different environments and target different audiences, which in turn leads to substantial differences in their characteristics. Grids comprise a manageable amount of different authorities, creating closed, moderate-sized communities called virtual organizations [6]. This allows for example

manual configuration of users and access rules for each authority. P2P systems consist of an arbitrary number of individuals without any common authority, thus requiring different approaches to trust and security. In a grid there are usually high-end resources available which allow a certain quality of service, while in a P2P system attempting to guarantee any quality of service is usually impeded by the dominant presence of low-end resources and high churn rates. In summary, grid computing addresses the matters of closed multi-organization groups with defined participants and authorities, whereas P2P computing addresses the matters of open individual groups with arbitrary participants and no authority. Some authors describe P2P computing as emerging variants of the grid principle [7] (for example referred to as ad hoc grid, personal grid, organic grid, desktop grid or public grid), while others consider that grids are in essence P2P systems [8]. There are several attempts to combine the ideas of grids and P2P systems [9] [10] [11] [12] and some attempts to pure P2P computing [13], but none of them deal with distributed cryptanalysis.

Existing frameworks for (distributed) cryptanalysis focus on a specific cryptanalytic challenge and do not follow the idea of an open, participant-driven system [3] [14] [15] [16].

III. TECHNICAL CHALLENGES

There are several undesired characteristics which are inherent to the peer-to-peer model. Opportunistic peers may try to gain an advantage over others, for example by exploiting foreign computational resources without sharing own resources. Worse, malicious peers may try to disrupt the operability of the system or particular peers for the sake of vandalism. Honest peers may be unreliable and can fail at any time with or without notice. Peers can enter and leave the system at any time with varying online session durations (*churn rate*). Network latency, available bandwidth and computational resources may be poor and vary as well. Connectivity of peers may be restricted by firewalls or NAT¹ routers. In summary, we classify the challenges for P2P-based cryptanalytic algorithms as follows:

- C1 Fault detection:** The system shall be able to detect incorrect intermediate results or incorrect operation of other peers' services. This includes the following faulty peer classes: malicious peers, opportunistic peers, unreliable peers.
- C2 Fault handling:** The system shall be able to handle failing peers, for example by cutting communication to peers which fall below a trust rating, and either recover from their faults (for example retry operation using another peer's service) or degrade gracefully (restrict or stop operation).
- C3 Offline capability:** As peers can go offline occasionally, it's desirable to not lose a task's progress made so far. The system shall allow peers to continue to work on a job even if going offline. Peers submitting a job shall be

¹Network Address Translation

able to retrieve the result even when they went offline for some time.

C4 Incentives: The participants shall be encouraged to contribute own resources. Technical measures shall prevent that opportunistic peers gain an unfair advantage over honest peers.

C5 Decentralized: As there is no provider, the system must organize itself and each service must operate in a decentralized manner. This requires a P2P network being able to bootstrap itself, traverse through NAT and provide scalable routing.

IV. DISTRIBUTION OF CRYPTANALYTIC JOBS

Whether a cryptanalytic job is applicable for a P2P-based distributed computation, depends primarily on the design of its algorithm. Therefore we first examine different algorithm design classes for their general purpose suitability for P2P-based computation. Then we discuss two practical examples.

In the following the term *job* refers to a certain computational challenge and *task* refers to a work unit, composing a slice of the overall job.

A. Algorithm Designs

An **exhaustive search** (or brute-force search) of the whole solution space is the most basic algorithm design. As every candidate solution can be calculated independently from each other, the exhaustive search job can be easily separated into parallel tasks. Typically a task consists of a set of continuous candidate solutions. Each task can be assigned to another peer and requires no interaction during computation (in the literature known as *embarrassingly parallel*). The result of a candidate solution can be a binary hit/miss or some kind of a cost function value. To avoid transmission of unnecessary data, the result of a task can be reduced to the s best candidate solutions.

A **divide and conquer** algorithm hierarchically divides a problem into subproblems and combines the results in the reverse order of the hierarchy. Subproblems on the same tier are disjoint and can be easily parallelized. The programming framework MapReduce [17] uses the divide and conquer design for cluster computing. However, care must be taken when trying to apply this design to P2P computing. Due to higher constraints on network resources, the distribution overhead of tasks must be in a reasonable proportion to the computation efforts. That is, the task at the bottom of the distribution hierarchy should be either a conventional non-dividing algorithm, or be divided and solved locally without further remote distribution. A variant of the divide and conquer design, called **decrease and conquer** reduces a problem into one smaller subproblem. As this is not parallelizable in its general form, decrease and conquer is not suitable for P2P computation.

The **dynamic programming** design makes use of the fact that certain problems consist of overlapping subproblems. The results of already solved subproblems thus can be used to speed up computation of other subproblems. This may or may

not be suitable for P2P computing, depending on 1) how large the overlappings and the saved computing times are, 2) how often the intermediate result data is updated and how large the updates are, and 3) if there are paths useful to be parallelized. In order to decrease computational redundancies and benefit from parallel computing, a mechanism is required to exchange the intermediate subproblem results. As peers have different resource capacities, it seems worthwhile to estimate whether a peer's contribution is reasonable depending on its available network bandwidth and computing power, and to adapt the peer's data exchange frequency as a trade-off between its available resources. Still, there may be dynamic programs which do not benefit from P2P computing at all.

A **backtracking** algorithm spans the solution space as a tree and uses depth-first search to find the correct solution. By partitioning the solution tree into subtrees, each subtree can be computed in parallel. This is for example used by the parallel backtracking framework BkFr [18]. However, similar to MapReduce mentioned above, this framework targets cluster computing and has been designed with a master/slave networking, thus being unsuitable for P2P computing. Intermediate results of a parallel backtracking task may help to reduce the solution space of other subtrees. If such information is used, this design corresponds to dynamic programming, and if not, it shares the characteristics of divide and conquer algorithms.

Instead of traversing through the whole solution space, **metaheuristic algorithms** attempt to converge to the correct or to a good enough solution with a specific strategy. For example **greedy algorithms** make choices on what seems to be the best solution of a subproblem, even when the subproblem has not been entirely solved yet. The (possibly approximate) results of already solved subproblems may be considered into the decision making of later subproblems, leading to similar distribution characteristics as dynamic programs. Many metaheuristics, like the **shotgun hill climbing** or **genetic algorithms**, use a random initial start condition and can run different solution attempts to find the best solution. Each attempt can be run in an *embarrassingly parallel* manner.

B. Example: Brute-force Attack on Symmetric-Key Cryptosystems

We now discuss how to perform a brute-force attack on a symmetric-key cryptosystem. This attack corresponds to the exhaustive search algorithm design and is therefore well suited for P2P-based computation. The set of all possible cryptographic keys of the cryptosystem spans the solution space. The solution space is divided into blocks, each block described as a range in the key space. The range may be variable to comply with the peers' different computing powers. This range is assigned to a peer as a task, together with a ciphertext to be broken. The computing peer attempts to decrypt the ciphertext with each key from the range. The success can be determined by comparing the decrypted plaintext to a known-plaintext, or by rating the decrypted plaintext with a cost function (for example to estimate the likelihood of the plaintext to be natural language). The task finishes either when being successful or

when the whole range has been processed. The peer then returns the aggregated results, receives credit and is ready to process the next task.

C. Example: Ciphertext-only Attack on Enigma

As another practical example we now examine how to break the Enigma with a P2P distributed ciphertext-only attack. The Enigma is a historical electro-mechanical cipher machine forming a symmetric-key cryptosystem [19]. The key space is determined by selection and mechanical configuration of several components: choice of rotors, their order and initial position; ring settings; plug connections; choice of reflector board (in later variants). There were several variants of the Enigma in use, all based on the same design but with different cryptographic key spaces. A common variant used by the German Wehrmacht in World War II had a key space of about 76 bit, later variants even more. Apart from partially-known-plaintext attacks (using *cribs*) there are ciphertext-only attacks which outperform the runtime of a brute-force attack. We now take a look how an attack technique proposed by Gillogly [20] can be P2P computed. The attack runs in three stages:

- 1) The first stage attempts to break the rotor settings without regard to the ring or plug settings (20 bit key space). In an exhaustive search each possible rotor setting is scored with an appropriate cost function to detect natural language, for example the index of coincidence. The best candidate solution is used as input to stage two.
- 2) The second stage determines the ring settings, again without regard to the plug settings (9 bit key space). The already small key space can be reduced by first analyzing one ring setting, then the next, leading to a practically negligible effort of 52 calculations. The candidate solution with rotor and ring settings is used as input to stage three.
- 3) In this stage a hill climbing algorithm is used to find the correct plug combination (47 bit key space). A single plug exchanges two letters, for example A-B, A-C, A-D and so on. Each correctly connected plug should improve the cost function value (which may be a different one than previously used, for example the sum of \log_2 trigram counts). If it did not, the algorithm removes the plug and tries the next plug connection. If a plug was found, the algorithm searches in a new round for the next plug. It stops, if the cost function score does not improve with an additional plug.

The Gillogly attack runs fast but suffers from rather low success rates, especially when the message is short or many plugs are used. As a simple improvement, instead of using one candidate solution the first stage can output the best s candidate solutions [21]. Stages two and three run for each of the s candidate solutions separately.

The exhaustive rotor search in stage one is embarrassingly parallel and can be easily P2P distributed (Fig. 1). Due to the low effort of the ring search in stage two, its computation can be combined to a common task together with the plug

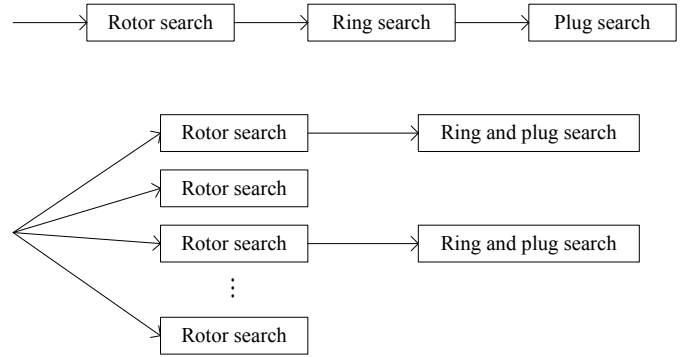


Fig. 1. Sequential and parallel Enigma attack

search. By using s candidate solutions there is no need to parallelize the dynamic approach of the hill climbing algorithm in stage three, instead, each of the s candidate solutions can be distributed in an embarrassingly parallel manner to another peer. To avoid idle peers in the transition from stage one to stages two and three, the best s currently known candidate solutions can be processed already in the next stages, even if the first stage may deliver better candidate solutions afterwards.

Sullivan and Weierud [22] presented further detail improvements to the hill climbing algorithm. In addition, the plug search runs for every possible rotor permutation combined with every possible setting of one ring (the other ring setting is disregarded). This multiplies the necessary computational effort in expectation for a further increased success rate. The M4 Message Breaking Project [16] used this concept together with distributed client/server computing in order to break historical messages encrypted with a four rotor Enigma. Sullivan and Weierud's approach corresponds effectively to our attack shown above, when s is set to the maximum amount of possible rotor (and ring) permutations, and is therefore still well suited for P2P distribution.

V. OUR APPROACH

Running a computational job in a P2P system requires a considerable amount of management work. A job must be sliced into smaller tasks. The tasks must be allocated to worker peers, and their status and progress needs to be tracked. Then the individual results must be collected, verified and merged to the overall solution of the job. The naïve attempt is to manage the distributed computation by a single peer. This composes a star-shaped management structure. As a single management peer obviously lacks scalability and reliability, in a P2P system the management responsibility should be shared between p peers. There are several possibilities to structure the management peers, for example as tree [10] [23], or as random structure of designated or randomly chosen managers [24] [13].

We propose a different approach without an explicit management role. Instead, the workers self-organize the computation of the job on top of a distributed storage. The idea is

to write the list of tasks and all task related information into the P2P storage. Any peer willing to participate in the job can choose an outstanding task, register itself as worker on this task and write the result when the task has been finished. Our approach focuses on embarrassingly parallel jobs.

A. System Model

The proposed approach poses a number of challenges itself, some of them being already solved by others, some being discussed below and some being beyond the scope of this paper. We assume, we have a P2P system based on a scalable network overlay (like Pastry [25] or CAN [26]). The system allows open access for new participants and each participant has a unique ID [27]. The ID does not change when a peer leaves and rejoins the system. The P2P system is able to pass messages between peers, even if some peers use NAT or block incoming connections, by using NAT traversal and relaying techniques. Message transport is secure on end-to-end level, i.e. the P2P system can guarantee authenticity and integrity of message transmission, as shown in [27]. It is not guaranteed that every peer is honest, thus a correctly transmitted and authentic message may still contain fraudulent promises. However we assume that the majority of participants is honest. Part of the P2P system is a distributed and reliable storage, for example a distributed hash table (DHT) with replication and caching.

We furthermore assume that there is some incentive for the peers to participate in distributed jobs. A peer participating in a job is a *worker*. An accounting system makes sure, that workers receive credits or a similar currency for the completion of tasks. Peers can quantify trust and reputation values for other known peers to weight probabilistic decisions but our approach does not strictly require the existence of a trust system.

B. Self-organizing Job Management

We propose to structure the task list as a tree which spans the search space of a job (Fig. 2). A leaf node of the tree represents an actual task, while an inner node represents an aggregation of the subtree's tasks. The tree is stored by the job submitter in the distributed storage and is accessible by all peers participating in the computation. The storage is evenly distributed among all participating peers, i.e. every peer stores a share of the job tree (see Section V-E below). In the following we refer to *peer* as participant of the system, and to *node* as entry of the job data structure which is saved in the storage.

The root node and each inner node can have up to c child nodes (e.g. $c = 2$). As the tree may become very large for computationally intensive jobs, it can be constructed on-the-fly while the job is being already computed on other parts of the tree. Each subtree pointer is either 1) empty and flagged as unfinished, as the child node has not been created yet, 2) points to a child node and is flagged as unfinished, or 3) points to a child node and is flagged as finished.

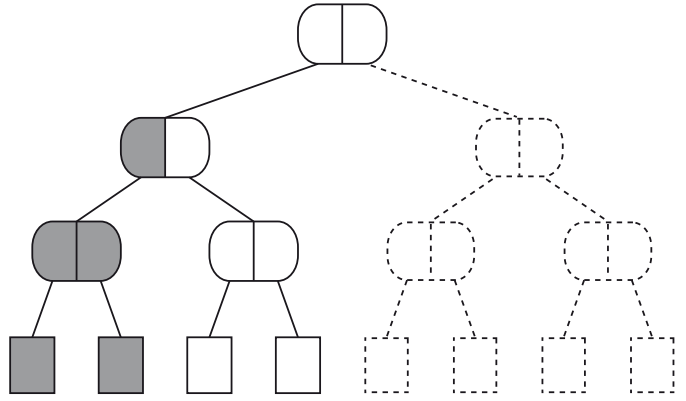


Fig. 2. Job tree with finished, unfinished and not yet created task nodes

A peer working on the job starts by reading the root node and traverses the tree, as follows: it enters an unfinished subtree and divides the search space where necessary, that is, creates a child node where the tree has not been fully constructed yet. Where the search space size matches a pre-defined value, the leaf level of the tree has been reached. The worker processes the computational task, writes the result data to this node and marks this subtree in the parent as finished. If all subtrees of a node are marked as finished, the worker retrieves the individual results from the child nodes, merges them, writes the merged result into the node and flags this node's tree in its parent as finished. These three steps – divide, calculate, and merge – are repeatedly performed until the root node contains the solution of the job.

We have to keep in mind that multiple peers are working on the tree in parallel. Hence, when a peer starts working on a task, it should mark the task as allocated. Furthermore the load on the storage should be kept low, in terms of both, query rate and data volume. High data volumes lead to higher bandwidth usages due to replication and churn handling. One measure to accomplish this is to remove finished subtrees and only to keep the merged results. To avoid querying the same occupied nodes over and over again, peers should traverse the tree in different paths. Fully randomized walks on the other hand would rapidly increase the tree size and put load on the storage as well. Instead a weighted probability with tendency in one direction seems favorable, for example depth-first traversal with frequent left turns.

The proposed approach shares the management burden between all peers participating in the job calculation. There is no need to select peers for a special management role. The job submitting peer merely needs to create the root node and can stay offline until it retrieves the solution.

Peers can become unavailable without notice anytime and never return a result to an allocated task. In a managed environment this failure type can be easily detected by requiring the computing peer to report regular status updates, or by actively requesting status updates. In our approach there is no manager involved who could determine whether a worker has timed out. If a worker peer claims an allocation of a task,

it's not feasible to infinitely block this task. The task allocation is therefore not an exclusive lock, but a recommendation for other peers to choose another task. This is especially important when the job is near completion, i.e. when all unfinished tasks are claimed to be allocated and there are still idle workers left. An idle worker may allocate an already taken task again in hope to process it faster and gain credit.

While there are enough unfinished tasks, a peer can always choose an unallocated task. Nevertheless, as described above, the job tree should not become too large as this puts excessive stress on the storage. Outdated allocations or allocations being obviously not beneficial due to other reasons (like low peer trust) can be ignored at the discretion of the worker peers.

C. Result Verification

When a peer writes the task result into the storage, there is no guarantee that the result is correct. The peer submitting a job certainly has only interest in correct results and therefore needs a way to verify them.

A common cryptanalytic task (like the symmetric-key brute-force attack) is to determine whether the solution being searched for is part of the solution subspace being processed by the peer, and what the solution value is. A false result falls therefore into one of the following failure subtypes: the peer claims to have found the solution, though this solution is actually wrong (*false positive*), or, the peer claims that the solution being searched for is not part of the solution subspace of the task, though the subspace actually contains the solution (*false negative*). The decision problem whether a false positive has been received, simply can be solved by repeating the computation for the claimed solution. Deciding about false negatives with absolute certainty is as hard as the task itself. The usual approach in P2P scenarios is to perform a voting on each task, that is, let one task being calculated by ≥ 2 peers and check whether the results are identical. This approach has the clear disadvantage of being wasteful. As trade-off between wasted resources and confidence voting could be performed on specific randomly chosen tasks. Thus, malicious peers will be detected probabilistically with reduced certainty.

As explained in Section IV there are cryptanalytic jobs, whose tasks do not deliver a hit/miss result, but an amount of solutions to subproblems which have been solved during computation of this task, for example the s best results. These s results can be verified by recalculation and comparison of their corresponding cost function score. Additionally, other random candidate solutions can be checked whether they actually score below the s claimed best results. Thereby, the correctness of the task result can be probabilistically checked – possibly more efficiently than redundantly repeating the computation of the task. The amount of candidate solutions being verified can be adjusted subject to a trust value assigned to the peer in question.

Let's assume we have an application whose most efficient verification is (probabilistic) voting or a similar computationally intensive method. The job submitter can assign a certain verification strategy to the job and demand, that the worker

performs one verification task for every k regular tasks in order to receive credits. Depending on the strategy, the worker selects the task to be verified randomly or, for example, focuses on untrusted peers. As there is no global view on trust, the actual decision lies with the worker choosing the task to be verified. The job submitter may also anytime choose to flag a finished task as to be calculated again by another peer (at the cost of additional credits), if the submitter does not trust the involved worker or workers.

One can assume that in a system where the majority of participants is honest, this majority will have an ambition to identify and exclude cheaters in order to not become a victim of cheaters itself. This implies that honest peers always perform the verification correctly and cheaters will be found with a certain probability.

In a more conservative system view one can assume that every honest peer will eventually become opportunistic, if this behaviour is not being sanctioned at all. This would imply, that more and more honest peers will decide to stop looking for cheaters. Instead an opportunistic peer will simply approve foreign task results without utilizing own resources, if there is no immediate personal drawback and no risk of being accused of bad behaviour. To avoid that a system of initially honest peers slowly converges to an unusable system state without an effective cheater detection, every rule infringement should be detectable and penalized (including defective verification). As honest peers have an interest in keeping the system in an operable state with correct results, the conservative assumption may be the worst case, which possibly proves wrong in practice. We will investigate in our future research which assumption is realistic and which measures must be taken to ensure a stable and operable system state.

D. Halt Condition and Subtree Cleanup

When the solution of a job has been found, it's reasonable to halt the execution and not further search through the solution space. The submitter can define the halt condition as part of the job description, for example when the cost function score is above a certain threshold. A worker claiming to have found the solution which matches the halt condition appends the claimed solution to the root node. As prematurely found solutions are always positive results, they can be easily verified by the participating workers. If the solution is correct, the peer will not select any additional tasks for calculation. If the solution turns out to be a false positive, the peer can consider the cheat attempt in his trust ratings and will continue working on the job as usual.

The verified removal of merged subtrees from the storage in an untrusted environment presents a challenge which cannot be solved satisfactorily with the mechanisms discussed above. The risk of losing major progress of a job requires an extremely high confidence in the merged results before deleting the partial results. Furthermore, the loss of the partial results circumvents a definite identification of the offender in case of cheating. Even a voting by multiple peers will usually not provide the required confidence. Hence the job submitter

must authorize the removal of subtrees itself, or delegate that responsibility to carefully selected fully trusted peers.

E. Storage

Our approach relies heavily on the P2P-distributed storage. The storage serves as an abstraction layer to encapsulate typical challenges not related to a specific application, like scalable data sharing. Each data node of the job tree structure is mapped onto a participating peer. This mapping happens via a cryptographic hash function and ensures a random, but even, distribution of the data among peers. As explained above, the job tree size grows with the number of concurrent workers. In a group with trusted peers, we therefore expect that the load per peer stays roughly constant and scales well with the number of participants. Without any absolutely trusted peers, the job submitter must authorize the removal of no longer needed data chunks from time to time. When the submitter does not do it, the storage load increases with the runtime of the job.

In addition to store and retrieve operations we require extra features from the storage which are not provided by a usual DHT. As some data entries will be queried more often than others, the storage may use adaptive replication to scale with the number of read operations, as seen in [28]. In our approach peers need to append their information to a task even if another peer claims an allocation to that task. Worker peers are only allowed to modify their own entries, but not to overwrite the entries of other peers. In order to avoid garbage, storage entries should dissolve if not being refreshed by the owner timely (*soft state*). The refresh interval shall adapt subject to 1) the type of data respectively the function the data is related to (for example computational results are more worth keeping than progress and allocation information), 2) the trustworthiness of the data owner (more likely to be meaningful information), and optionally 3) the system load (an idle system can tolerate potentially outdated or useless information more easily than a system at its capacity limits).

In summary, we can state the following storage requirements:

- S1** Provide an access model in which every participant is able to read other peers' data, but is only allowed to modify his own data.
- S2** Prevent unauthorized modifications, at least with a high probability [29].
- S3** Each DHT entry shall have a configurable durability.

VI. SUMMARY AND OUTLOOK

We presented an approach allowing the user to run distributed cryptanalysis without relying on third party service providers. The P2P technique unburdens the user from configuring and administrating a dedicated network server, but leads to several challenges. We discussed which algorithm designs are suitable for P2P-based cryptanalysis and proposed a new fully decentralized approach. Our presented approach is scalable, fully self-organizing and copes with possible wrong results from unreliable or untrusted peers.

Our next step will be the implementation and detailed evaluation of our approach as presented in this paper. As a proof of concept, we already incorporated the P2P middleware of the peers@play project [30] into the existing KeySearcher plugin from CrypTool 2 [31]. CrypTool 2 is a high-level cryptographic toolkit, allowing the user to experiment with cryptographic functions and cryptanalytic algorithms in a graphical editor. By P2P-enabling CrypTool 2, users can easily build some cryptanalytic jobs and run them highly distributed on a large P2P system. For extending the KeySearcher plugin, we used the simple approach with a manager and any number of workers to perform a brute-force attack on symmetric-key cryptosystems. We demonstrated this plugin at CeBIT 2010 [32]. Furthermore, we are going to investigate how to distribute sophisticated cryptanalytic algorithms like the quadratic sieve algorithm in our P2P-based system.

REFERENCES

- [1] D. P. Anderson, "Boinc: A system for public-resource computing and storage," in *GRID '04: Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing*. Washington, DC, USA: IEEE Computer Society, 2004, pp. 4–10.
- [2] J. N. Silva, L. Veiga, and P. Ferreira, "nuboinc: Boinc extensions for community cycle sharing," in *Proceedings of the 2008 Second IEEE International Conference on Self-Adaptive and Self-Organizing Systems Workshops*. Washington, DC, USA: IEEE Computer Society, 2008, pp. 248–253. [Online]. Available: <http://portal.acm.org/citation.cfm?id=1524875.1524997>
- [3] "distributed.net," Published on the WWW at <http://www.distributed.net>.
- [4] L. M. Vaquero, L. Rodero-Merino, J. Caceres, and M. Lindner, "A break in the clouds: towards a cloud definition," *SIGCOMM Comput. Commun. Rev.*, vol. 39, pp. 50–55, 2009.
- [5] I. T. Foster, "The anatomy of the grid: Enabling scalable virtual organizations," in *Euro-Par '01: Proceedings of the 7th International Euro-Par Conference Manchester on Parallel Processing*. London, UK: Springer-Verlag, 2001, pp. 1–4.
- [6] I. Foster and A. Iamnitchi, "On death, taxes, and the convergence of peer-to-peer and grid computing," in *In 2nd International Workshop on Peer-to-Peer Systems (IPTPS'03)*, 2003, pp. 118–128.
- [7] H. Kurdi, M. Li, and H. Al-Raweshidy, "A classification of emerging and traditional grid systems," *IEEE Distributed Systems Online*, vol. 9, no. 3, p. 1, 2008.
- [8] D. Talia and P. Trunfio, "Toward a synergy between p2p and grids," *IEEE Internet Computing*, vol. 7, pp. 96, 94–95, 2003.
- [9] F. Cappello, S. Djilali, G. Fedak, T. Hault, F. Magniette, V. Néri, and O. Lodygensky, "Computing on large-scale distributed systems: Xtrem web architecture, programming models, security, tests and convergence with grid," *Future Gener. Comput. Syst.*, vol. 21, no. 3, pp. 417–437, 2005.
- [10] A. J. Chakravarti, G. Baumgartner, and M. Lauria, "Self-organizing scheduling on the organic grid," *IJHPCA*, vol. 20, no. 1, pp. 115–130, 2006. [Online]. Available: <http://dblp.uni-trier.de/db/journals/ijhPCA/ijhPCA20.html#ChakravartiBL06>
- [11] D. Caromel, A. d. Costanzo, and C. Mathieu, "Peer-to-peer for computational grids: mixing clusters and desktop machines," *Parallel Comput.*, vol. 33, no. 4-5, pp. 275–288, 2007, proActive Framework.
- [12] K. Bhatia, "Peer-to-peer requirements on the open grid services architecture framework," OGSA-P2P Research Group, Tech. Rep., 2005.
- [13] J. Verbeke, N. Nadgir, G. Ruetsch, and I. Sharapov, "Framework for peer-to-peer distributed computing in a heterogeneous, decentralized environment," in *In Proceedings of the 3rd International Workshop on Grid Computing*. Springer-Verlag, 2002, pp. 1–12.
- [14] A. M. de Oliveira Candia, "Flexible cryptanalysis in java," *The International Journal of Forensic Computer Science*, vol. 1, no. 1, 2006.
- [15] P. Majkowski, M. Rawski, T. Wojciechowski, Z. Kotulski, and M. Wojtynski, "Heterogenic distributed system for cryptanalysis of elliptic curve based cryptosystems," in *ICSENG '08: Proceedings of the 2008 19th International Conference on Systems Engineering*. Washington, DC, USA: IEEE Computer Society, 2008, pp. 300–305.

- [16] S. Krah, "M4 message breaking project," Published on the WWW at http://www.byteref.org/m4_project.html, (retrieved on 2010-04-20).
- [17] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," in *OSDI'04: Proceedings of the 6th conference on Symposium on Operating Systems Design & Implementation*. Berkeley, CA, USA: USENIX Association, 2004.
- [18] M. Kouril and J. L. Paul, "A parallel backtracking framework (bkfr) for single and multiple clusters," in *CF '04: Proceedings of the 1st conference on Computing frontiers*. New York, NY, USA: ACM, 2004, pp. 302–312.
- [19] R. A. Miller, "The cryptographic mathematics of enigma," *Cryptologia*, vol. 19, pp. 65–80, 1995.
- [20] J. J. Gillogly, "Ciphertext-only cryptanalysis of enigma," *Cryptologia*, vol. 19, pp. 405–413, 1995.
- [21] H. Williams, "Applying statistical language recognition techniques in the ciphertext-only cryptanalysis of enigma," *Cryptologia*, vol. 24, pp. 4–17, 2000.
- [22] G. Sullivan and F. Weierud, "Breaking german army ciphers," *Cryptologia*, vol. 29, pp. 193–232, 2005.
- [23] D. Castellà, I. Barri, J. Rius, F. Giné, F. Solsona, and F. Guirado, "Codip2p: A peer-to-peer architecture for sharing computing resources," in *Advances in Soft Computing*, 2008, pp. 293–333.
- [24] N. Andrade, W. Cirne, F. Brasileiro, and P. Roisenberg, "Ourgrid: An approach to easily assemble grids with equitable resource sharing," 2003.
- [25] A. Rowstron and P. Druschel, "Pastry: Scalable, distributed object location for routing for large-scale peer-to-peer systems," in *Proceedings IFIP/ACM Middleware 2001*, Heidelberg, Germany, November 2001.
- [26] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, "A scalable content-addressable network," in *Proceedings of the 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM)*. San Diego, CA, USA: ACM Press, 2001, pp. 161–172.
- [27] A. Wacker, G. Schiele, S. Schuster, and T. Weis, "Towards an authentication service for peer-to-peer based massively multiuser virtual environments," *Int. J. Advanced Media and Communications*, 2008.
- [28] M. Knoll, H. Abbadi, and T. Weis, "Replication in peer-to-peer systems," in *International Workshop on Self-Organizing Systems (IWSOS'08)*, Vienna, Austria, December 2008.
- [29] S. Schuster, A. Wacker, and T. Weis, "Fighting cheating in p2p-based mmvcs with disjoint path routing," *Electronic Communications of the EASST*, vol. 17, 2009.
- [30] University of Duisburg-Essen and University Mannheim, "peers@play homepage," Published on the WWW at <http://www.peers-at-play.org/>, (retrieved on 2010-04-20).
- [31] "CrypTool 2," Published on the WWW at <http://www.cryptool2.vs.uni-due.de/>, (retrieved on 2010-04-20).
- [32] University of Duisburg-Essen, "CeBIT 2010: Sicherheit und virtuelle Welten," Published on the WWW at <http://www.uni-due.de/ssc/messen/rexp.php?EID=262>, March 2010.