

Analysis-Resistant Malware

John Bethencourt*

University of California, Berkeley / Carnegie Mellon University

Dawn Song†

Brent Waters‡

SRI International

Abstract

Traditionally, techniques for computing on encrypted data have been proposed with privacy preserving applications in mind. Several current cryptosystems support a homomorphic operation, allowing simple computations to be performed using encrypted values. This is sufficient to realize several useful applications, including schemes for electronic voting [16, 12, 17] and single server private information retrieval (PIR) [19, 10]. In this paper, we explore an alternative application for these techniques in an unexpected setting: malware. The possibility of malicious uses for PIR and general techniques for computing on encrypted data was first suggested by Young and Yung [32]. Counter-intuitively, these techniques enable malware which renders some aspects of its behavior provably resistant to forensic analysis, even with full control over the malware code, its input, and its execution environment. While methods for general purpose computation on encrypted data have not yet been realized, we explore the potential use of current techniques. Specifically, we extend the earlier work of Young and Yung by investigating in depth the possibility of malware which employs private stream searching (a recent, more flexible variant of PIR) techniques to find and retrieve specific pieces of sensitive information from compromised hosts while hiding its search criteria. Through an evaluation of the goals of attackers and the constraints under which they operate, we determine that PIR techniques are an attractive technology to malware authors with the potential to increase the threat of targeted espionage. We go on to demonstrate the present feasibility of PIR-based malware through a series of experiments with a full implementation of a recent private stream searching scheme. Through the example of PIR-based malware, we highlight the more general possibilities of computing on encrypted data in a malicious setting.

1 Introduction

Malware analysis is an important process which can help guide an appropriate response to a security breach or reveal the motivations of the malware author. Currently, malware authors employ a host of methods to frustrate analysis, thereby extending the malware lifespan and concealing their aims. These analysis resistance techniques include code obfuscation, self-checking and self-modifying code, polymorphism, and metamorphism [11, 21]. A number of worms specifically attempt to detect the presence of debugging tools and alter or terminate operation [30]. These techniques have triggered an arms race between increasingly powerful analysis and reverse engineering tools [30] and ever more clever techniques on the part of malware authors.

Despite the sophistication exhibited by many pieces of recent malware, theoretical results suggest that malware authors are fighting a losing battle in this arms race. Secure obfuscation for general programs is not possible due (at least) to contrived classes of programs that are impossible to obfuscate [2], and recent results have further shown that many natural, interesting classes of programs are also impossible to obfuscate [14].

However, an alternative approach exists for malware to hide certain aspects of its behavior. Several related cryptographic notions known variously as public key program obfuscation [22] and cryptocomputing [29] concern the transformation of a program into an “encrypted” representation that provably hides the function it computes while still allowing execution of the program. The key difference in this model is that the output of the encrypted program is unintelligible to the party executing the program, and can only be transformed into the actual output with the help of an auxiliary private key (kept by the originator of the program). The negative results on program obfuscation do not apply in this case, since they require an obfuscated program to produce the same output as the original.

Unlike current methods [11] for program obfuscation, which at best delay analysis, schemes within the public key obfuscation model allow provable security. While general purpose public key program obfuscation is an unsolved problem and current methods for cryptocomputing are only

*Supported by a US DoD NDSEG Fellowship and NSF CNS-0716199.

†Supported by NSF CNS-0716199.

‡Supported by NSF CNS-0524252, CNS-0716199, and the US Army Research Office under the CyberTA Grant No. W911NF-06-1-0316.

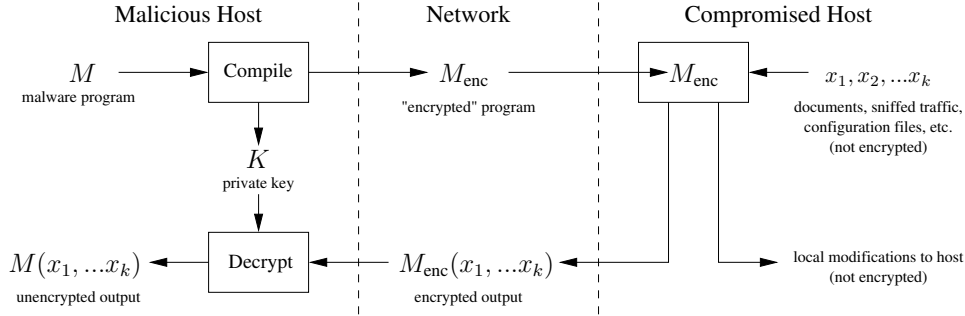


Figure 1. Malicious usage of public key program obfuscation.

effective for small circuits, efficient solutions to more specialized problems are available. In particular, a number of schemes based on homomorphic encryption have been proposed for the problem of single server private information retrieval (PIR), which may be viewed as a special case of public key obfuscation. A PIR scheme allows a client to retrieve an entry x_i from a database (x_1, x_2, \dots, x_n) on an untrusted server while preventing the server from learning which entry i they retrieved. This may be trivially accomplished by sending the entire database to the client, but PIR schemes generally provide reduced communication complexity (e.g., $O(\sqrt{n})$ or $O(\log n)$ rather than $O(n)$). Further extensions and variations allow keyword based search and retrieval of documents [9] and operation within a streaming model [22]. From the malware author’s perspective, such schemes may be useful for retrieving sensitive documents or system information by keyword while hiding the search criteria. In this case, the malware author would play the role of the client, and the compromised host would act as the server.

This possible use of PIR by malware was first considered by Young and Yung, who also went on to point out that other techniques within the framework of cryptocomputing may be useful to malware authors [32]. In much earlier work, they considered simpler techniques with similar motivation [31].

Contributions. In this paper, we further explore the general threat of public key program obfuscation or cryptocomputing techniques employed in a malicious setting, considering in particular malware employing a private stream searching scheme as an example of currently feasible techniques within this model. Specifically, we provide the following contributions.

- An investigation of the applications of private stream searching in malware.
- A detailed evaluation of the goals of targeted malware authors which suggests that such malware is attractive

for malicious purposes.

- The first publicly released implementation of a private stream searching system and experimental results demonstrating its use is currently feasible.
- Some brief additional results on the potential for distributed use of PIR by worms.

Organization. In Section 2 we discuss the general cryptographic framework and tools available to a malware author, including public key obfuscation, cryptocomputing, homomorphic encryption, and PIR. In Section 3 we explore the goals of the malware author and evaluate the utility of PIR-based malware in particular, and in Section 4 we give experimental results demonstrating the present technical feasibility of PIR-based malware. In Section 5 we give a high-level discussion of the implications of these possibilities and countermeasures before concluding in Section 6. Related work is given throughout, especially in Section 2. We also give consideration to an additional, more speculative scenario for malicious use of PIR in Appendix C; reading this section is not essential for understanding the rest of the paper.

2 Analysis Resistance via Cryptography

We now survey several cryptographic definitions and tools and discuss how they may be used by malware authors interested in preventing analysis of their malware.

2.1 Framework: Public Key Obfuscation

When defining the problem of private stream searching, Ostrovsky and Skeith also introduced the more general problem of public key program obfuscation [22]. Informally, we may consider a scheme for public key obfuscation to consist of a space of relevant programs \mathcal{C} and two probabilistic algorithms `Compile` and `Decrypt`. The `Compile` algorithm takes a program $A \in \mathcal{C}$ and returns an “encrypted”

version A_{enc} along with a private key K . The Decrypt algorithm processes an output from the encrypted program using the private key to “decrypt” the output. We require two properties:

Correctness

Let $A \in \mathcal{C}$ and $(A_{\text{enc}}, K) = \text{Compile}(A)$. Then for all x , we require that $\text{Decrypt}(A_{\text{enc}}(x), K) = A(x)$.

Hiding

In the absence of K , A_{enc} should reveal nothing about A beyond that it is in \mathcal{C} . More precisely, we define a game between an adversary B and a challenger C .

1. B chooses two programs $A_0, A_1 \in \mathcal{C}$ and sends them to C .
2. C flips a coin $b \in \{0, 1\}$, computes $(A_{\text{enc}}, K) = \text{Compile}(A_b)$, and sends A_{enc} to B .
3. B outputs a guess b' .

We define the advantage of an adversary B as a function of the security parameter k (elsewhere omitted from the notation) to be $\text{Adv}_B(k) = |\Pr(b = b') - \frac{1}{2}|$. We require that $\text{Adv}_B(k)$ be a negligible function for all PPT’s B .

Note that in order for the hiding property to be satisfied, it is essential that Compile be a probabilistic algorithm with many potential encrypted representations for each program. For more rigorous definitions the reader should refer to [22]; these intuitive notions will suffice for our purposes. The interested reader may also wish to review the largely equivalent notion of cryptocomputing [29].

Example. This framework is depicted in a malicious context in Figure 1. Here, $M \in \mathcal{C}$ is a piece of malware that is encrypted to produce M_{enc} , which is run on a compromised machine. The algorithm Compile hides certain characteristics of M ; namely, those that distinguish it from other members of \mathcal{C} . However, in the process the output is rendered unusable. This output $M_{\text{enc}}(x_1, \dots, x_n)$ must now be returned to the malware author and given to Decrypt along with K before the actual output $M(x_1, \dots, x_n)$ may be discerned. Note that M_{enc} may also be permitted to produce some unencrypted outputs, provided every other member of \mathcal{C} produces them in the exact same way.

As an example, one may imagine M to be a host-based vulnerability scanner. The program M inspects various aspects of the host’s configuration, then processes this information according to a list of rules for detecting various software and configuration vulnerabilities, ultimately producing a concise summary of the resulting discoveries. In this case, the malware author may wish to hide the scanning criteria and vulnerability detection logic to ensure the continued secrecy of valuable 0-day vulnerabilities.

Discussion. It is important to remain clear on the relationship between \mathcal{C} , the output and behavior of M_{enc} , and precisely which characteristics of M are hidden. In the example of the vulnerability scanner, an analyst observing M_{enc} will certainly see which data x_1, \dots, x_n it reads on the remote host. The best we can do, then, is to take \mathcal{C} to be the set of programs which read this data, perform some computation, and return resulting values (of a particular size) over the the network, and then we will hide the nature of the computation which M performs. This ability could be surprisingly useful. If x_1, \dots, x_n is an extremely large, general set of inputs (e.g., the contents of all program binaries, libraries, the kernel, configuration files, and sniffed network traffic), the malware analyst observing M_{enc} will have essentially no information about the vulnerabilities the malware may have been looking for. If M had run, however, the analyst would find the same vulnerabilities on the system that M had, and be able to respond by disabling the affected services until they can be patched. In general, one should keep in mind that it will not be possible to hide any characteristics of the malware that inherently must affect its control flow or output that is not to be returned to the author (i.e., local changes to the compromised host).

Another point to understand is that a scheme for public key obfuscation provides malware with genuinely new abilities, beyond what is possible through other approaches. One may imagine malware taking the more simplistic approach of reading its inputs and performing the necessary computations in the clear, then encrypting the output with an embedded public key before sending it over the network. The source of the malware (who has the corresponding private key) would be able to decrypt the returned output. In this case, the output of the malware would be hidden from anyone monitoring the network. However, this approach does nothing to prevent an analyst with access to the host on which the malware is running from observing what it is computing and sending back. A malware author who intends to hide the information they seek and how it is to be derived from data on the compromised host must assume that the malware’s code and execution environment will be analyzed upon detection.

2.2 Present Techniques: Homomorphic Encryption

Public key obfuscation schemes for fully general classes of programs \mathcal{C} do not yet exist. However, methods are available for more specialized classes. We now discuss these methods and what may be accomplished using them.

Essentially all work in this framework so far has been based upon the use of *homomorphic cryptosystems*. Suppose for some public key cryptosystem we have a space of plaintexts P , a space of ciphertexts C , and an encryption

function (for a particular public key) $E : P \rightarrow C$. Then we say that the cryptosystem supports a homomorphism $f : P^n \rightarrow P$ if there exists an operation $f' : C^n \rightarrow C$ such that

$$f'(E(x_1), E(x_2), \dots, E(x_n)) = E(f(x_1, x_2, \dots, x_n))$$

for all $x_1, \dots, x_n \in P$. More precisely, we should say that

$$D(f'(E(x_1), E(x_2), \dots, E(x_n))) = f(x_1, x_2, \dots, x_n) ,$$

since E will typically be a probabilistic function. Here $D : C \rightarrow P$ is the private decryption algorithm corresponding to E .

Such a homomorphism allows one to perform the operation f on encrypted values and obtain an encryption of the result, thus enabling computation on encrypted data. This provides the essential building block for realizing schemes in the framework of public key obfuscation. Generally speaking, the algorithm M_{enc} may encrypt its input values with an embedded public key, and then perform its computations by using an operation $f' : C^k \rightarrow C$ on them along with (already encrypted) embedded constants.

Several cryptosystems support a single homomorphic group operation, including ElGamal [13], Goldwasser-Micali [15], and Paillier [25]. The more sophisticated cryptosystem of Boneh, Goh, and Nissim supports arbitrary additions of plaintexts and a single multiplication [3]. Unfortunately, no known cryptosystem supports homomorphic operations that are sufficient to realize general computation on encrypted data [24]; finding such a cryptosystem or demonstrating that they do not exist is a long standing and important open problem. A notable partial exception is the scheme of Sander, Young, and Yung, which supports both boolean OR and NOT, which are sufficient for general computation [29]. However, in this scheme the ciphertext size doubles after every operation, so only small numbers of operations are feasible.

Cryptosystems supporting a single homomorphic group operation are, however, sufficient for a number of useful applications. In particular, they are sufficient to solve the problem of private information retrieval. As an example, we give in Appendix A a simple PIR scheme with $O(\sqrt{n})$ communication complexity using a generic construction given in [23], instantiated with the Paillier cryptosystem. The example serves to illustrate the usage of homomorphic encryption and shares the flavor of more advanced constructions for PIR. Readers that have not previously seen a construction for PIR may find it enlightening. The example also provides a more concrete illustration of the definitions for public key program obfuscation.

More sophisticated approaches to PIR allow search based on keywords rather than array indices [9] and operation on a sequence of documents with no a priori

bound [22], dubbed private stream searching. Throughout the rest of the paper, we will generally assume the use of a scheme for keyword based private stream searching, as this variant of PIR offers more flexibility. A query will then take the form of a list of keywords rather than an index i , and we will assume that it can be matched against an arbitrary number of documents one by one, updating a fixed length, encrypted buffer of current results after each.

2.3 Properties Offered

In both the general case of public key obfuscation and the specific problem of PIR, it is possible to trivially achieve the hiding property by simply retrieving the entire set of inputs x_1, x_2, \dots, x_n and running the original program M locally on the machine of the malware author. The only useful schemes for public key obfuscation are then those that reduce the communication to something closer to the size of the actual output $M(x_1, \dots, x_n)$. What such a scheme offers, then, is the combination of two properties: low communication and program hiding. Either one may be trivially achieved alone.

These properties have another interesting relationship. When a malware author wishes to hide the function being computed on the compromised host, this actually intensifies the need for the reduced communication in the following way. Whenever either a public key obfuscation scheme or the trivial method of returning all the input data is being used to hide the function being computed, all an analyst on the remote host will be able to see is the set of inputs that are read. The malware author will want this set to be as large and generic as possible to minimize the information revealed about their activities. In the case of PIR in particular, the set of inputs read reveals everything the malware *could* possibly be retrieving, so it must be large if meaningful secrecy is to be achieved. This relationship between the need for secrecy of the function computed (or documents returned in the special case of PIR) and the need to read a great deal of data on the remote host narrows the circumstances under which the trivial approach of returning all input is possible. In particular, the malware author may wish to limit exfiltration bandwidth used by the malware to reduce the chances of detection.

The two properties offered by these techniques are then

Low bandwidth

The malware may scan a large amount of data and thus effectively hide its intentions, but only use bandwidth to send back what is deemed relevant, thus decreasing the likelihood of detection.

Hiding

In case of discovery, the malware will not reveal what specific information was sought. Even with full access to and control over the malware binary (or even its

source code), its execution environment, and all data included with it, security professionals and researchers will be provably unable to determine which specific pieces of data the malware was retrieving.

This raises a natural question: “Under what circumstances are these properties important to a malware author?”. To answer this question, in the next section we explore the goals of authors targeted malware and the constraints under which they operate.

3 PIR-Based Malware

Having considered the general definitions of public key obfuscation and cryptographic tools for implementing it, we now consider in depth a specific example of such techniques that could be used in malware today. Specifically, we investigate the possibility of malware employing PIR techniques.

3.1 Targeted Espionage

Compromised hosts are of course desired for a variety of purposes, including DDOS attacks and as stepping stones for further malicious activities. PIR techniques, however, will be naturally most useful in the case of malware designed to retrieve information. Recent years have seen increasing cases of malware found within organizations specifically targeted for military or industrial espionage.

In one such case, dubbed “Trojagate”, ten’s of thousands of commercially sensitive documents were captured by malware on hosts within dozens of prominent Israeli companies and exfiltrated to about 100 receiving servers, causing widespread media attention [26, 28]. The trojan, known as Rona or Hotword.B, was specifically written for espionage purposes and had not been previously encountered in the wild. Furthermore, the incident was not an opportunistic attempt on the part of an isolated hacker to obtain valuable information; instead the malware was sold to and used by several private investigation firms that had been hired by three of Israel’s top telecom companies. The trojan was introduced into the targeted organizations through carefully executed social engineering efforts employing infected documents attached to emails and delivered on CD’s. There it used lists of keywords to trigger keystroke logging and screen captures, in addition to searching for sensitive documents [20]. Due to the low profile maintained by the infected machines, the trojan was not discovered for over a year and a half, causing what the head of the Israeli investigation called “one of the gravest scandals in ... industrial and market espionage in Israel”. The incident ultimately resulted in stock losses, numerous arrests, and a possible attempted homicide.

A number of other incidents highlight the threat of carefully targeted malware. In 2004, several New York banks

were affected by a piece of malware that was designed to infect only specific systems within their organizations; a project by the Ponemon Institute revealed malware that searched for documents flagged as confidential or “critical”; and anti-virus firm MessageLabs discovered a trojan specifically designed to obtain data from an application used in airplane design – suggesting military espionage [26]. These incidents reveal a setting in which stealthy operation may be of paramount importance to the malware author; explicitly searching for sensitive documents may unacceptably reveal a link between the malware and its origin when it is eventually analyzed. In cases of military, political, or commercial espionage, PIR techniques may be the key to effectively obtaining sensitive information while exfiltrating minimal data and thus avoiding detection.

3.2 Attacker Goals and PIR

We now consider in greater detail the possible motivations of malware authors and challenges they face to determine the conditions under which PIR techniques will be beneficial. Table 1 lists a number of types of information they may seek from a compromised host.

Exfiltration strategies. For each example in Table 1, we give the general type of document or kind of data desired (second column), the criteria for finding the specific documents or pieces of data of interest (third and fourth columns), and the risk incurred by performing an explicit search for the items of interest (fifth column). In the sixth column we give the bandwidth necessary to exfiltrate all data of that type, and in the seventh, the bandwidth necessary to exfiltrate only a specific item of interest. Note that some of these examples are concerned with transient data (e.g., network traffic), which must be recorded by malware present on the system as it arises, while others correspond to static data which is stored on the host. In the case of static data, the malware may immediately begin scanning for and exfiltrating data upon arrival and terminate upon completion, while malware seeking transient data must wait within the system.

To retrieve the desired information, a malware author has three options.

Return all

Exfiltrate all data of that type.

Explicit search

Include keyword list or other search criteria, and only return the relevant items.

Private search

Use PIR techniques to return the relevant items while hiding the search criteria.

Information desired	Data to return	Data to search	Search query	Importance of query secrecy	Bandwidth for all data of type	Bandwidth for desired data	Utility of PIR
System passwords	Logged keystrokes	Keystrokes, window titles and content	Trigger text indicating password entry	Low	< 10 KB per day	10's of bytes per password	Low: use <i>explicit search</i> or <i>return all</i>
Bank and other online account credentials	HTTP POST request content	Destination URL	List of domains (financial, etc.)	Low	< 60 KB per day [7]	< 3 KB per post [7]	Low: use <i>explicit search</i> or <i>return all</i>
User web activity	URL's in browser history and pages in cache	URL's, web page text	List of keywords and URL's of interest	Potentially high	100's of MB to multiple GB ¹	10's of KB per page	High
Business materials	Productivity application documents (i.e., .doc, .xls, etc.)	Document content	Keywords of interest	Potentially high	10's of MB	100's of KB per document	High
Visual snapshot of user activities	Screenshots	Keystrokes, window titles and content	Trigger text and keywords of interest	Potentially high	100's of MB per day ²	≈100 KB per screenshot	High
SIP / VoIP conversations (to and from one phone)	Speech recording	Name of caller or callee, text from voice recognition	List of names, keywords of interest	Likely high	≈100 MB per day ³	≈3 KB per minute	High
Email (to and from one user)	Email headers and body	Email addresses, email body	List of addresses, keywords of interest	Likely high	100's of KB per day	10's of KB per document	High

Table 1. Example scenarios for the capture and exfiltration of sensitive information by malware. The first column lists a general type of information a malware author or user may wish to obtain from a compromised machine. Columns 2 - 5 describe the specific pieces of data to be retrieved and how the malware may search for them. Columns 6 and 7 estimate bandwidth necessary to exfiltrate all data of that type or only the pieces of interest, and the final column suggests the resulting utility of PIR techniques. Bandwidths given are rough estimations (to an order of magnitude) of typical usage, and compression is assumed where possible.

¹By default, Internet Explorer sets the size of the web cache to 10% of the installed hard drive space, often causing unreasonably large caches. Other browsers use more modest values.

²One screenshot every 2 to 10 seconds of user activity, with 3 to 7 hours of user activity per day.

³About 30 to 40 minutes per day of G.729 or G.723.1 encoded speech.

The **return all** strategy may be employed whenever a type of data is small enough to be exfiltrated in its entirety without arousing suspicion or the malware author knows that no system will be monitoring bandwidth. In this case, PIR techniques are not necessary. Otherwise, the malware author will need to selectively return only items matching a list of keywords or other criteria. If these keywords or criteria do not reveal an unacceptable link with the source of the malware or their intentions, the search may be performed normally within the malware (**explicit search**), and PIR techniques are again unnecessary. However, when exfiltrating all data possibly of interest would consume a conspicuous amount of bandwidth and revealing the specific information sought would be unacceptable, the **private search**

strategy becomes key to achieving the malware author's goals. We will now consider the bandwidth constraints of malware authors and their motivation for hiding the information they seek in order to determine when both these conditions hold.

Available bandwidth. To date, malware authors have primarily retrieved information from compromised machines by directly forming new outgoing connections rather than attempting to piggy back data on existing traffic using network covert channels. To help avoid detection, the traffic may be minimally disguised as legitimate, for example by using port 80 and formatting the traffic as an HTTP request.

Recently, a web proxy dubbed Web Tap that attempts

to detect automated outbound transmissions disguised as browsing sessions was developed [7]. By recording statistics such as the timing and sizes of HTTP requests in legitimate browsing sessions, Web Tap was able to detect a number of spyware clients and backdoors that tunnel communication in this manner. To avoid detection in the place of similar monitoring techniques, the malware must time traffic to blend in with existing web browsing sessions and throttle its bandwidth to be below alert thresholds. Both of these techniques have been observed in the wild [8, 20]. During the development of Web Tap, detailed statistics on normal web browsing traffic patterns were recorded in order to set the alert thresholds on outbound traffic. The results suggest thresholds of about 60 KB per user per day, of which at most 20 KB may be directed to a single receiving server. Lower thresholds result in an unmanageable amount of false positives. These thresholds provide the malware author with a very clear bound on the possible rate of covert exfiltration from a user workstation. Referring to Table 1, we see that in all but one or two of the listed scenarios malware attempting to exfiltrate all data of a particular type would result in a high risk of detection through bandwidth monitoring techniques. An exception is keystroke logging data (first row); an entire day’s worth of logged keystrokes could likely be exfiltrated undetected. Exfiltrating all outgoing user HTTP traffic may also be possible (second row), depending on the amount of activity and particular threshold levels. In all other considered cases, a malware author concerned with detection would need to employ either an **explicit search** or a **private search** for the specific data of interest if bandwidth monitoring may be present.

Query secrecy. We now evaluate the need for malware authors and users to conceal the criteria used to conduct a search. While this is a more subjective task, considering each of the listed scenarios provides some insight. In each case, by employing a private search, the malware author would only reveal that the type of information it seeks is that of the second column. In contrast, by searching explicitly, they would reveal which specific pieces of information they wish to obtain.

The first two rows correspond to general attempts to obtain account credentials. In these cases, the malware author is likely to have little need to conceal the specific accounts they hope to access. The remaining scenarios correspond to more insidious attempts to gather information that suggest a program of focused espionage, as exemplified by the real life anecdotes in Section 3.1. In these cases, a query for specific information is likely to be highly sensitive to the investigation. This is especially true of the scenarios of the last two rows, which correspond to monitoring of personal communications.

Solution. In summary, it is clear that in many situations – especially those motivated by attempts at targeted espionage – the malware author has little bandwidth available for exfiltration of sensitive data, yet must not reveal the specific information they seek. PIR is key to achieving the attacker’s goals in these situations. To gauge the immediacy of the threat of malware employing PIR techniques, we need to evaluate the communication and computational overhead incurred and the logistical hurdles to using them in practice. In the following sections, we give a description of our full implementation of a recent scheme, our adaptation of it to data exfiltration, and the results of experiments demonstrating that it can be used in malware today.

4 Implementation and Experiments

4.1 Implementation

We have built a complete toolkit (“privss”) implementing a recent private stream searching scheme [6] and made it available on the web under the GPL [5]. Of course, our intention in making it publicly available is not to reduce the work of malware authors; rather the toolkit is provided in the hope that it will be useful for privacy preserving applications⁴ and to other security researchers. Those interested in additional information on the toolkit may find more technical details in Appendix B.

Adaptation to email exfiltration. To evaluate the logistical hurdles a malware author may face in using private streaming searching within malware, we adapted the `privss` package to process and exfiltrate email, as suggested in the last row of Table 1. Although any kind of data may be exfiltrated, email is a typical example of a sensitive document type, roughly similar in size and quantity to productivity application documents, web pages, etc.

For each message, a set of associated keywords is extracted from the message headers and body. To allow case-insensitive matching, all extracted words are converted to lowercase. The keyword matching method of [6] results in some (low) probability of each word appearing in a document causing a “false positive” match, in which the document is returned (consuming space in the fixed length results buffer) despite not matching the actual query. This probability depends on the size allocated for the encrypted query, and may be reduced with larger queries. Since grammatical words and other generic English words are unlikely to form useful queries, a list of the 1,500 most common English words (derived from the British National Corpus [1])

⁴Notwithstanding processing time versus communication time arguments [27], private stream searching may be useful in applications in which bandwidth is limited by cost or other constraints.

is used to filter the extracted keywords. This has the important effect of reducing these false positives. This also reduces query secrecy in the sense that it reveals that the query does not include any of those words, but a brief review of the words present in the filtering list makes it clear that none are likely to be useful search terms in any case. Finally, gzip compression is applied to reduce the length of the message before processing it with the private search algorithm.

A key task in any use of a system for private stream searching is the selection of a bound on the number of documents to be retrieved. The fixed length of the buffer that is incrementally updated by the private search algorithm during document processing allows the possibility of an “overflow” of matches, resulting in the inability to later reconstruct the matching documents. The need for a fixed length buffer is actually inherent in the problem of private stream searching. Allowing the algorithm to grow the buffer as necessary would require it to distinguish matching documents from non-matching documents, in violation of the security definition for private stream searching. Thus any secure scheme must ensure that this is not possible, only allowing fixed length buffers.

The malware author then needs some a priori information about the number of documents that may match their query and their total size. In some search and exfiltration applications (e.g., as in the third row of Table 1), this is easily obtained, but in most cases they will need to estimate or limit the total number of documents searched and estimate the portion likely to contain their keywords. For the purposes of email search and exfiltration, this may be aided by a period of initial monitoring. The malware may be designed to spend an initial period recording statistics on the number of emails sent to and received by the user and their average length, and compute a size for its buffers accordingly. This is the strategy employed by our adaptation of the `privss` package.

4.2 Experiments

Having put together a system for searching and exfiltrating email, we ran a series of tests to evaluate the communication overhead incurred and other factors affecting the practicality of these techniques for the malware author. The experiments were conducted on a dataset of about 200,000 emails sent within the Enron corporation from 1999 through 2002 that was publicly released in 2003 as a part of the investigation by the US Federal Energy Regulatory Commission [18]. By using a dataset already publicly released, we gain the advantage of a large volume of real documents (including many on sensitive topics) without raising privacy concerns.

Two basic scenarios were considered in this context. In

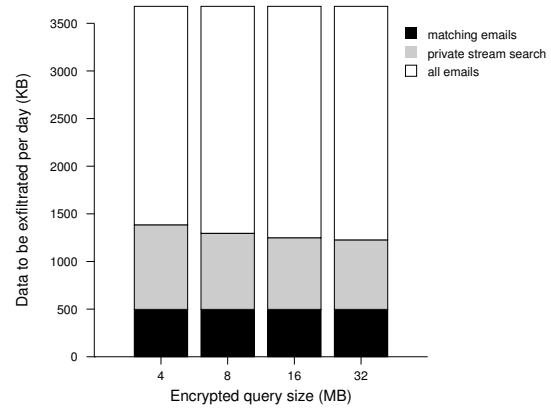
the first case, we imagine that the workstation of a single user has been compromised with malware that will monitor all messages received by or sent by that individual. This scenario allows us to consider the use of private stream searching malware in a setting with relatively few documents and more strict requirements on acceptable exfiltration bandwidth. At the other extreme, we also consider the case of a compromised mail server that will spy on all mail passing through its MTA. In this case, a large volume of documents will be searched, and more bandwidth may be used to surreptitiously exfiltrate results.

Compromised workstation. To analyze the case of the compromise of a single user’s workstation, we collected all email from the Enron corpus to and from a single user over a period of ten days. The private search and exfiltration system scanned previous email to determine the average number of messages seen per day and their average size, allowing it to pick sizes for the buffers kept during the search. While general statistics such as these may be used by the malware to configure search parameters, the keywords used in the query of course may not be considered without compromising secrecy. Buffer space was allocated to allow up to ten total messages of average size to be retrieved over the ten day period, or an average of one per day. Given these parameters, we ran a search using the keyword “sensitive”. This was repeated for various sizes of the encrypted query hash table. Using a larger encrypted query has the effect of reducing the false positive rate and results in the malware accordingly selecting a somewhat smaller results buffer. This procedure was further repeated for three different users, namely, Richard Shapiro and James Steffes (VP’s in Enron) and Jeff Dasovich (Director of Enron). These were the three users most well represented in the corpus and thus provided a volume of mail most similar to typical usage.

Averaging the three trials for each query size produced the results displayed in Figure 2(a). The black bar displays the daily bandwidth required to directly transmit the (compressed) messages which match the query, as when searching explicitly. The gray bar gives the daily bandwidth used by the private search, and the white bar gives the daily bandwidth that would be necessary to return all mail observed. The figure also displays per user exfiltration detection thresholds typical of software designed to detect such activity. Specifically, the upper (60 KB) and lower (20 KB) lines correspond to the detection thresholds determined by Web Tap [7] for the total daily bandwidth and the total bandwidth leaving for any one site. Judging from the figure, in this scenario, retrieving all mail observed on the workstation in the presence of bandwidth monitoring is not possible without detection. Using an encrypted query of 8 MB or more allows one to exfiltrate the results of a private



(a) Single user workstation.



(b) Mail server.

Figure 2. Results of email exfiltration experiments.

search to a single external site, while using an encrypted query of 2 - 4 MB may require the data to be split among sites. Nevertheless, in all cases the private search is feasible relative to the total bandwidth threshold. It is also apparent that the private search uses two to three times the bandwidth necessary to retrieve the files with an explicit search. This overhead is incurred in two ways: the a priori fixed size of the results buffer,⁵ and cryptographic overhead.

Each email required about 200 - 300 milliseconds⁶ to be processed. Each user sent and received an average of 51 messages per day, thus requiring only about 10 to 15 seconds of processor time per day overall. This result strongly suggests that computational overhead is not likely to give away the presence of malware in this scenario.

A somewhat more important consideration for the malware author is pushing a 2 - 16 MB encrypted query to the compromised host while avoiding detection. This is not likely to pose much difficulty, however. Surreptitiously *infiltrating* significant amounts of data to a compromised host is far easier than exfiltrating data, due to the lopsided bandwidth usage of users under normal circumstances. The piece of malware which initially infects the system may retrieve an encrypted query with a series of HTTP requests back to a machine controlled by the author. Each retrieved piece may be disguised as a media file such as an image or video. Given the current popularity of online video websites such as YouTube, one such request may even suffice. Although encrypted queries larger than 16 MB could be used, there is little to gain from doing so as a 16 MB query virtually eliminates false positives for searches of this scale.

⁵Up to 30 matching messages were allowed between the three users tested, but a total of 19 messages matched.

⁶The processor in the workstation used was a 64-bit, 3.2 GHz Pentium 4. The workstation had 2GB of RAM, although memory capacity did not play a significant role in this experiment.

Conversely, the number of false positives resulting from a query much smaller than 2 MB may be problematically large, but infiltrating at least 2 MB should pose no difficulty.

Compromised mail server. Now we turn to the scenario of a compromised mail server monitoring all mail passing through its MTA. The private search and exfiltration system was invoked with the search keyword “sensitive” as before, this time processing one day’s worth of mail in the corpus to and from all users (approximately 2,500 messages). The buffer sizes were initialized to the same bounds as in the previous case, scaled up in proportion to the volume of mail processed. As before, the experiments were repeated for several query sizes. The results are shown in Figure 2(b). In this case, we do not have clear bounds on possible detection thresholds, but some observations can be made. Naively exfiltrating all mail observed will (at least) double outbound bandwidth, almost certainly causing an alert if the server’s bandwidth is monitored. A malware author attempting to retrieve specific messages passing through the server while concealing which they are seeking would do well to employ a private search. The cost incurred in this scenario is the usage of approximately twice the bandwidth of an explicit search.

While infiltrating the encrypted query will likely not pose any more of a problem than in the previous scenario, the computational costs may be troublesome to the malware author in this case. While the processing time (on the same machine as in the last example) remains at about 200 - 300 milliseconds per message, the CPU usage patterns may be somewhat more predictable on a server machine than a workstation. Since the normal time required for the MTA to process a message will be far less than 200 milliseconds, the malware author will have to take some care to not dramati-

cally alter the load on the machine in a way that may alert a host-based intrusion detection system or an observant administrator. Of course, mail may be queued for processing by the private searching code and processed whenever it is convenient. One possible strategy for obscuring the source of load would be to inject the relevant code into spam filtering software, which in many cases requires over a second to process each message. Instances of malware injecting code into existing libraries and running processes to disguise the source of load and for other reasons have been observed in the wild. In the case of this experiment, the malware would need to hide about 8 - 13 minutes of additional CPU usage per day.

Summary. In short, private stream searching appears to be an entirely effective method for malware to surreptitiously search and exfiltrate email. Malware designed to save and return messages on a specific sensitive topic will be able to do so without revealing the topic of interest upon analysis; all that will be determined is that it scans email in general. Furthermore, as our implementation demonstrates, there is nothing to prevent these techniques from being used immediately. This example of PIR-based malware illustrates the more general possibility of malware employing public key obfuscation techniques to hide its behavior, and thus the intentions of its author.

5 Discussion

Evaluating the threats highlighted by this paper at a high level, the primary concern is that, in the short term, PIR techniques will encourage more bold use of malware in obtaining sensitive information. While these methods do not allow malware authors to retrieve any data they otherwise could not, they reduce the risk in doing so. The scandal resulting from the “Trojagate” incident was devastating to the Israeli telecom companies and private investigators responsible, and the possibility of this kind of fallout serves as a useful deterrent to similar illicit activities. Private searching and other private information retrieval techniques may unfortunately reduce this deterrent. Looking farther into the future, if and when more advanced schemes are developed within the framework of public key obfuscation, they will also enter the malware author’s toolbox.

While little can be done to directly address the possibility of the use of such techniques in malware, it is helpful to at least be aware that it is not always possible to determine precisely what malware may be computing or exfiltrating. Instead, when analyzing malware one must assume that in principle it could be retrieving any data that could be derived from anything it has read. As for more specific methods for detecting and preventing this threat, there are

several directions worth further consideration. First, the significant computation required by methods for computing on encrypted data may increase the vulnerability of this type of malware to host based anomaly detection systems. This is especially true in the case of servers, in which the CPU load may be more predictable.

6 Conclusions

In summary, an evaluation of the goals of malware authors and the risks they face in retrieval of sensitive information reveals that PIR may prove to be an attractive technology for the next generation of malware. By minimizing the bandwidth necessary to exfiltrate the desired data while hiding precisely what is sought, PIR techniques allow the malware author or user to simultaneously reduce the risk of detection and the risk of association with the malware in case of its analysis. This new threat raises the challenge of finding better methods for detecting and preventing these techniques. Looking farther ahead, PIR techniques may be the first of a series of new methods for analysis-resistance in malware.

Acknowledgements

The authors would like to thank Jason Franklin for several essential suggestions in preparing this work. We would also like to thank Moti Yung for pointing out important, closely related work of which we were previously unaware.

References

- [1] The british national corpus. Oxford University Computing Services. Information available at <http://www.natcorp.ox.ac.uk/>.
- [2] B. Barak, O. Goldreich, R. Impagliazzo, S. Rudich, A. Sahai, S. Vadhan, and K. Yang. On the (im)possibility of obfuscating programs. *International Cryptology Conference*, 2001.
- [3] D. Boneh, E.-J. Goh, and K. Nissim. Evaluating 2-dnf formulas on ciphertexts. *Theory of Cryptography Conference (TCC)*, 2005.
- [4] J. Bethencourt. The libpaillier library. Available at <http://acsc.csl.sri.com/libpaillier/>.
- [5] J. Bethencourt and B. Waters. The privss toolkit. Available at <http://acsc.csl.sri.com/privss/>.
- [6] J. Bethencourt, D. Song, and B. Waters. New techniques for private stream searching. 2006. Extended abstract appeared in the *IEEE Symposium on Security and Privacy*, full version available at <http://www.cs.cmu.edu/~bethenco/search.ps>.

- [7] K. Borders and A. Prakash. Web tap: Detecting covert web traffic. *ACM Conference on Computer and Communications Security*, Vijayalakshmi Atluri, Birgit Pfizmann, and Patrick Drew McDaniel, eds., ACM, Washington, D.C., October 2004.
- [8] K. Borders, X. Zhao, and A. Prakash. Siren: Detecting evasive malware (short paper). *IEEE Symposium on Security and Privacy*, 2006.
- [9] B. Chor, N. Gilboa, and M. Naor. Private information retrieval by keywords. Department of Computer Science, Technion, Technical Report CS0917, 1998.
- [10] C. Cachin, S. Micali, and M. Stadler. Computationally private information retrieval with polylogarithmic communication. *Eurocrypt*, 1999.
- [11] C. Collberg, C. Thomborson, and D. Low. A taxonomy of obfuscating transformations. Department of Computer Sciences, The University of Auckland, Technical Report 148, July 1997.
- [12] I. Damgård and M. Jurik. A generalisation, a simplification and some applications of paillier’s probabilistic public-key system. *International Workshop on Practice and Theory in Public Key Cryptography (PKC)*, 2001.
- [13] T. ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *International Cryptology Conference*, August 1984.
- [14] S. Goldwasser and Y. T. Kalai. On the impossibility of obfuscation with auxiliary input. *Symposium on Foundations of Computer Science*, Pittsburgh, Pennsylvania, October 2005.
- [15] S. Goldwasser and S. Micali. Probabilistic encryption. *Journal of Computer and System Sciences*, 28(2), April 1984.
- [16] M. Hirt and K. Sako. Efficient receipt-free voting based on homomorphic encryption. *Eurocrypt*, 2000.
- [17] A. Kiayias and M. Yung. The vector-ballot e-voting approach. *Financial Cryptography*, 2004.
- [18] B. Klimt and Y. Yang. Introducing the enron corpus. *Conference on Email and Anti-Spam (CEAS)*, Corpus available at <http://www.cs.cmu.edu/~enron/>, 2004.
- [19] E. Kushilevitz and R. Ostrovsky. Replication is not needed: Single database, computationally-private information retrieval. *Symposium on Foundations of Computer Science*, Miami Beach, Florida, October 1997.
- [20] R. Murawski. Data exfiltration techniques: How attackers steal your sensitive data. *Virus Bulletin Conference*, October 2006.
- [21] J. Newsome, B. Karp, and D. Song. Polygraph: Automatically generating signatures for polymorphic worms. *IEEE Symposium on Security and Privacy*, May 2005.
- [22] R. Ostrovsky and W. Skeith. Private searching on streaming data. *International Cryptology Conference*, 2005.
- [23] R. Ostrovsky and W. E. Skeith, III. A survey of single database pir: Techniques and applications. Cryptology ePrint Archive, Report 2007/059, 2007. <http://eprint.iacr.org/>.
- [24] R. Ostrovsky and W. E. Skeith, III. Algebraic lower bounds for computing on encrypted data. Cryptology ePrint Archive, Report 2007/064, 2007. <http://eprint.iacr.org/>.
- [25] P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. *Eurocrypt*, 1999.
- [26] B. Sullivan. Israel espionage case points to new net threat. *MSNBC News*, See <http://www.msnbc.msn.com/id/8145520/>, June 2005.
- [27] R. Sion and B. Carbunar. On the computational practicality of private information retrieval. *Network and Distributed System Security Symposium*, 2007.
- [28] R. Singer. Top-tier israeli firms suspected of spying on competition. *Haaretz Daily News, English Edition*, May 2005.
- [29] T. Sander, A. Young, and M. Yung. Non-interactive cryptocomputing for NC^1 . *Symposium on Foundations of Computer Science*, New York, New York, October 1999.
- [30] A. Vasudevan and R. Yerraballi. Cobra: Fine-grained malware analysis using stealth localized-executions. *IEEE Symposium on Security and Privacy*, 2006.
- [31] A. Young and M. Yung. Deniable password snatching: On the possibility of evasive electronic espionage. *IEEE Symposium on Security and Privacy*, 1997.
- [32] A. Young and M. Yung. *Malicious cryptography: Exposing cryptovirology*. Wiley, 2004.

A Example PIR Scheme

Here we give a very simple example of a private information retrieval scheme (from [23]) constructed from a homomorphic cryptosystem, placed in the framework of general public key obfuscation. We use the Paillier cryptosystem [25], which supports an additive homomorphism via multiplication of ciphertexts. That is, $\forall x_1, x_2 \in P$, $D(E(x_1) \cdot E(x_2)) = x_1 + x_2$.

Suppose the PIR server stores n database entries, each considered to a single bit for simplicity. Assume the values are arranged in a square matrix $X = (x_{ij})_{1 \leq i, j \leq \sqrt{n}}$. Now, in the context of public key obfuscation, we are considering the class of programs \mathcal{C} that read all entries in the database and return the entry at some predetermined index. Then Compile and Decrypt may operate as follows:

Compile(M) $\rightarrow M_{\text{enc}}, K$

Let i', j' be the index of the bit that M returns. Generate a Paillier key pair with private key K_{priv} . Next, compute the vector $Q = (q_i)_{1 \leq i \leq \sqrt{n}}$, where

$$q_i = \begin{cases} E(1) & \text{if } i = i' \\ E(0) & \text{otherwise.} \end{cases}$$

Note that Paillier is a probabilistic cryptosystem, so in general each q_i is distinct. Now let $K = (K_{\text{priv}}, j')$ and define M_{enc} as follows:

```

hosta$ privss-qcon illuminati mkultra
hosta$ ls
enc_query  prv_key
hostb$ ls
enc_query  kennedy.jpg  report.pdf  interview.mp3
hostb$ privss-search enc_query enc_res kennedy.jpg    rfk "robert kennedy"
hostb$ privss-search enc_query enc_res report.pdf     mkultra "sodium pentothal"
hostb$ privss-search enc_query enc_res interview.mp3  sirhan illuminati
hostb$ ls
enc_query  enc_res  kennedy.jpg  report.pdf  interview.mp3
hosta$ ls
enc_query  enc_res  prv_key
hosta$ privss-recon enc_query enc_res prv_key
hosta$ ls
enc_query  enc_res  prv_key  report.pdf  interview.mp3

```

Figure 3. Example usage session with the privss toolkit.

$$M_{\text{enc}}(X) \rightarrow R$$

For each $j \in \{1, \dots, \sqrt{n}\}$, compute $r_j = \prod_{i=1}^{\sqrt{n}} q_i^{x_{ij}}$. Output $R = (r_1, r_2, \dots, r_{\sqrt{n}})$.

Decrypt($M_{\text{enc}}(X) = R, K = (K_{\text{priv}}, j')$) $\rightarrow \{0, 1\}$

Using K_{priv} , decrypt $r_{j'}$ and output the result.

To see that Decrypt will produce the correct output, note that by the homomorphism

$$D(r_{j'}) = \sum_{i=1}^{\sqrt{n}} x_{ij'} D(q_i) = x_{ij'} .$$

The hiding property is achieved directly from the semantic security of Paillier encryption, which has in turn been proven based on the decisional composite residuosity assumption (DCRA).

With $O(\sqrt{n})$ communication, this simple PIR scheme is inefficient relative to modern schemes. However, it serves to illustrate the usage of homomorphic encryption and shares the general flavor of more advanced schemes.

B The privss Toolkit

The privss toolkit is a general purpose package for practical usage of a recent private stream searching scheme. It utilizes a library implementing the Paillier cryptosystem, which we have also made available [4]. A number of extensions to the basic scheme described in [6] are also implemented, including the Bloom filter-based index storage, the technique for reducing the size of the Bloom filter, and the method for transparently handling files of arbitrary length. The interface of the toolkit is designed for straightforward invocation by larger systems in addition to manual usage. It provides three command line tools. The functionality of these tools mirrors the three algorithms described

in [6]: QueryConstruction, StreamSearch, and FileReconstruction.

privss-qcon

Generates an encrypted query and private key for the specified keywords using the QueryConstruction algorithm.

privss-search

Processes a file using an encrypted query, creating or updating a buffer of results according to the StreamSearch algorithm.

privss-recon

Using the private key and a buffer from privss-search, recovers the files which matched the query using the FileReconstruction algorithm.

In the framework of public key obfuscation as described in Section 2, privss-qcon implements the Compile algorithm, privss-search and the encrypted query would be bundled together to form M_{enc} , and privss-recon implements the Decrypt algorithm.

Figure 3 depicts a simple example usage session of the privss toolkit. First, an encrypted query for the keywords “illuminati” and “mkultra” is generated on Host A. The file enc_query is sent to Host B, where it is used to process three files, each of which has a list of associated keywords. The file enc_res is produced. Back on Host A, it is used with the private key prv_key to reconstruct the files with keywords matching the query. Note that the privss-search tool does not attempt to read keywords directly from the files. Instead it allows the user (or higher-level invoking application) to specify keywords explicitly; in this way a variety of document types may be handled in application specific ways. In this example, keywords for the latter two files may have been obtained using the pdftotext and id3info programs.

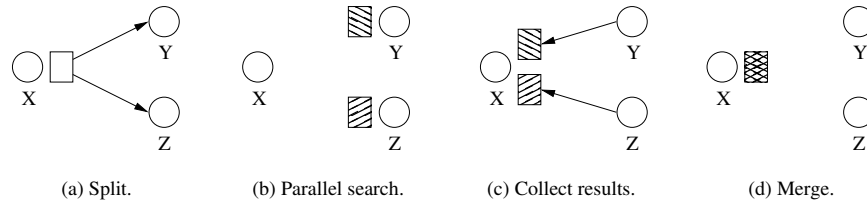


Figure 4. A distributed private search. Hosts Y and Z search two sets of documents in parallel, then host X combines the results.

C Distributed Searches and Worms

Here we consider an additional, somewhat more speculative usage of PIR techniques in malware: distributed searches. If a malware author is seeking a particularly rare piece of data across a large number of hosts, receiving results buffers from each individually will incur a large amount of wasted bandwidth on the receiving host. Since the receiving host will likely also be a compromised machine, this will increase the chances of detection and failure to obtain the desired information.

However, a particular technical property of both private stream searching schemes to date [22, 6] allows an alternative approach. After a buffer of encrypted results has been initialized for subsequent use with a particular query, the buffer may be split by simply producing any number of copies of it. These may be sent to multiple hosts, where the `StreamSearch` algorithm (see Appendix B) may be employed on each to process documents in parallel. Eventually, the resulting buffers (with contents now diverged from one another) may be merged back together into a single buffer of the same size. The resulting buffer may be used with the `FileReconstruction` algorithm to obtain the matching documents from *all* hosts, just as if the documents on each host had been processed one after another with a single buffer. In short, due to the homomorphism which originally allows the scheme to function, merging buffers is possible by simply multiplying the contents of the buffers together element by element (with minor additional considerations [6]). This process for distributed private searches is depicted in Figure 4. In step (a), a host X initializes a buffer for the `StreamSearch` algorithm and sends copies to hosts Y and Z. Hosts Y and Z each search their own set of documents in step (b), before returning their copies of the buffer to X in step (c). As step (d) host X applies the homomorphism to obtain a single buffer containing the results from both Y and Z. At this point, host X may continue the search by processing its own documents with the buffer. Note that this process may be applied recursively; host Y for example can in turn pass the buffer on to further hosts and merge the results between steps (a) and (b). This pattern of splitting and merging behavior suggests the possibility of

a private search being conducted by a worm in a tree like fashion across a very large number of hosts. While this is a highly speculative scenario, we now give rough calculations evaluating the feasibility of an extreme example that may be suited to such a distributed search.

Suppose an attacker wishes to find the PGP / GPG private key of a specific individual and furthermore does not wish to reveal their interest in that individual. Although it’s not entirely clear when this secrecy would be essential (perhaps an investigation of a particularly elusive criminal), we continue the example due to technical interest. The attacker assumes the key is stored on some workstation used by the user,⁷ but does not know the location of the machine or does not wish to specifically connect to it, thereby revealing their intentions. Therefore they decide instead to release a worm which will attempt to recover the key using the distributed searching technique shown in Figure 4.

In this scenario, the search could be accomplished while consuming relatively little bandwidth to and from any single host. Suppose one million hosts will be infected by the worm in all, and assume each host has at most one stored private key to be searched. Using the scheme of [6] with a 256 KB query, under 1000 false positives should result, and a results buffer of about 256 KB should suffice to ensure overflow does not occur. We omit the details of these calculations here for brevity; for similar calculations see [6]. Suppose the worm, carrying the encrypted query within it, infects new hosts in a tree pattern with a branching factor of k using a precomputed hitlist. In this phase, each host will receive the 256 KB encrypted query, and send it out to each subsequent host it infects. Eventually, each leaf host runs the `StreamSearch` algorithm on any stored keys discovered, and return its 256 KB results buffer to the host that infected it. Each non-leaf receives k buffers, merges them, and recursively returns the result. Thus, overall, each host uses $(k + 1) \cdot 256$ KB of outbound bandwidth. With $k = 3$, for example, each host would generate a total of 1 MB of outbound bandwidth, and the infection tree would have a height of 13. While it is unclear how likely such an attack is in practice, this type of widespread, distributed private

⁷Of course, it would most likely be encrypted with a passphrase, but after retrieval it could be subjected to an offline dictionary attack.

search forms an intriguing possibility that perhaps deserves further attention.