PRIVATE SEARCHING FOR SENSITIVE FILE SIGNATURES

John Solis
Scalable and Secure Systems Research, Sandia National Labs, Livermore, CA, U.S.A.

Keywords: Private searching, Private matching, Homomorphic encryption applications.

Abstract: We consider the problem of privately searching for sensitive or classified file signatures on an untrusted server. Inspired by the private stream searching system of Ostrovsky and Skeith, we propose a new scheme optimized for matching individual file signatures (versus keyword matching in documents). Our optimization stems from the simple observation that a complete list of matching file signatures can be replaced by a much smaller encrypted bitmask. This approach reduces a server's response overhead from being linear in the number of matched documents to linear with respect to a system robustness parameter.

1 INTRODUCTION AND MOTIVATION

Government organizations are responsible for protecting sensitive and classified information from unauthorized disclosure. A potential solution to this problem is to augment existing virus/malware scanners with classified file signatures. During a routine scan, any computer discovered to contain classified files can be immediately confiscated. Unfortunately, this approach leaves the augmented databases vulnerable to local exploits: databases may be leaked by new malware that compromise the computer. This is problematic since an adversary can use the database to verify classification status of arbitrary files.

The ideal solution would be a scanner, capable of executing on untrusted computers, that searches for classified signatures without revealing any information. In particular, the host itself should not learn about the signatures being searched or when they have been located.

We propose a new method for privately detecting classified file signatures on untrusted systems. Inspired by existing private stream searching systems, we use Paillier encryption to construct a simple bitmask identifying all classified signatures present on a particular host. No information can be leaked since all operations are over encrypted ciphertexts – even in compromised or untrusted contexts.

2 RELATED WORK

The closest related work is the Ostrovsky-Skeith private stream searching system (Ostrovsky and Skeith, 2007). It allows a client to privately search through a stream of documents, located on separate server, and retain copies of any document containing any combination of secret keywords. The server to client communication complexity is bounded by $O(m * log\ m)$, where m is the maximum number of documents that can be retrieved. Subsequent work (Bethencourt et al., 2009), improves communication and storage complexity to $O(m)$.

In our scenario, we are primarily interested in the *existence* of a file, not necessarily its content. Although testing for existence can be performed by matching file contents, this requires a high communication overhead (especially for large files).

Private Set Intersection (PSI) (Freedman et al., 2004) allows two parties, each containing a private set of inputs, to jointly calculate their intersection without leaking extra information about either set. In our scenario, the sets would be (1) sensitive/classified file signatures and (2) public file signatures. The intersection, i.e., all identified classified files, can be sent to a central server for processing. Our goal is to develop a scheme with lower communication complexity than the approaches discussed here.

Solis J..
PRIVATE SEARCHING FOR SENSITIVE FILE SIGNATURES.
DOI: 10.5220/0003466703410344
In *Proceedings of the International Conference on Security and Cryptography* (SECRYPT-2011), pages 341-344
ISBN: 978-989-8425-71-3

3 PRIVATE SIGNATURE SEARCHING SCHEME

Problem Statement. The *administrator* of a large organization wants to scan its computers, referred to hereafter as *servers*, in search of sensitive files. An efficient private signature searching scheme should not reveal any information to the server about the signatures being searched or when they have been located.

Solution Overview. Building such a scheme requires a solution with (1) minimal communication complexity, and (2) a privacy-preserving method for identifying matching file signatures.

We accomplish the first requirement by observing that, for each file signature in the administrator's classified database, we are only interested in communicating a single bit of information: *does this signature exist on the server?* To query for the entire database, we construct a simple bitmask where individual bits correspond to specific signatures. The exact one-to-one mapping from sensitive file signatures to specific bitmask indices is known only to the administrator.

For the second requirement, we apply the homomorphic properties of the semantically secure Paillier encryption system (Paillier, 1999) to our bitmask. The server can manipulate the bitmask because, in the Paillier system, multiplying two ciphertexts together results in an encryption of the sum of the plaintexts: $\mathcal{E}(\alpha) * \mathcal{E}(\beta) = \mathcal{E}(\alpha+\beta)$. Plaintexts are represented as elements of $\mathbb{Z}_n$ and ciphertexts in $\mathbb{Z}_{n^2}$, where $n = pq$ is an RSA number with $p < q$ and $p \nmid (q-1)$.

Now assume the administrator provides the server with a set of ciphertexts of the form $\mathcal{E}(2^i)$, and for a given set of ciphertexts, each value of i is used only once. To discover sensitive files the server computes the signature of all files it contains and multiplies (in an oblivious manner) into our encrypted bitmask: $\mathcal{E}(2^i)$ when the signature is in the classified set, or $\mathcal{E}(0)$ when it is not.

Since each ciphertext is of the form $\mathcal{E}(2^i)$, the product of all ciphertexts is essentially computing the binary XOR over the original plaintexts. Giving a simple example: $\mathcal{E}(8) * \mathcal{E}(4) = \mathcal{E}(12)$. In binary: $\mathcal{E}(1000) * \mathcal{E}(0100) = \mathcal{E}(1100)$, i.e., binary XOR. The administrator decrypts the final encrypted bitmask and uses the private one-to-one mapping to determine matching signatures.

3.1 Formal Construction

Let $\mathcal{E}(\cdot)$ denote the Paillier encryption function, $\mathcal{C}$ denote an ordered set of classified signatures, $\mathcal{F}$ denote the set of file signatures on a server, $\mathcal{Q}$ denote a set of encrypted bitmasks, and $\mathcal{H}_i : \{0,1\}^* \rightarrow \{0,1\}^h$ denote a one-way cryptographic hash function that maps arbitrary length input strings to strings of bit-length h.

A private signature searching scheme is a tuple of algorithms:

Administrator:KeyGen(τ)**.** The administrator executes the key generation algorithm of the Paillier cryptosystem with security parameter τ to find an appropriate RSA number, $n = pq$. To guarantee that elements of $\mathcal{Q}$ are correctly represented (i.e., a unit mod n), select an m such that $2^m < min\{p,q\}$. Output the Paillier public key $PK = n$, corresponding secret key $SK = \{p,q\}$, and maximum supported classified signature set size m.

Administator:Setup$(SK, \mathcal{C}, \mathcal{F}, k)$**.** On input of a classified signatures set and file signatures set, the administrator verifies $|\mathcal{C}| \leq m$ and $m < \sqrt{|\mathcal{F}|}$ and aborts if either test fails. Otherwise, construct a set of k one-to-one mappings from elements in $\mathcal{C}$ to specific bit positions in our bitmask as follows:

Since $\mathcal{C}$ is an ordered set, we take the existing position of $c_i \in \mathcal{C}$ and use it as as the corresponding bit position in our bitmask, e.g., the third element c_3 maps to 2^3. Now let $\mathcal{K}$ be a set of k values selected uniformly at random from $\mathbb{Z}_n$. Each key, along with a (keyed) pseudo-random permutation function $PRP_k(\mathcal{C})$, generates a unique permutation of $\mathcal{C}$ and unique mappings from elements to bit positions. Update the secret key to include the set of permutation keys, i.e., $SK = \{\{p,q\}, \mathcal{K}\}$.

Next, compute a set of tables with encrypted values representing the individual bitmask bits. For each $t \in \{1,...,k\}$ and each $k' \in \mathcal{K}$, initialize each table, $\mathcal{D}_t$, with $s = 2|\mathcal{F}|$ entries of $\mathcal{E}(0)$. Select an index i uniformly at random and for the j-th element $c_j \in PRP_{k'}(\mathcal{C})$, set $\mathcal{D}_t[\mathcal{H}_i(c_j) \bmod s] := \mathcal{E}(2^j)$. If a collision occurs between any two elements of $\mathcal{C}$, the entire array is discarded and a new index chosen for $\mathcal{H}$. Repeat the process until no collisions have occurred and store the index in set $\mathcal{I}$. Note that the $m \leq \sqrt{|\mathcal{F}|}$ requirement guarantees the probability of a collision will always be less than $\frac{1}{2}$ (see Choice of m discussion in Section 4 below).

The final step is to compute k tables, $\mathcal{W}$, with s entries of $\mathcal{E}(0)$ in each table. These tables are used by the **Scan** algorithm as "working" tables to store intermediary results.

Server:Scan$(PK, \{\mathcal{D}, \mathcal{W}, \mathcal{I}\}, \mathcal{F})$**.** This algorithm outputs a single encryption element representing all classified signatures present on a server.

For each $f \in \mathcal{F}$ and each $i \in \mathcal{I}$, let t be the index of i in $\mathcal{I}$ and set $\mathcal{W}_t[\mathcal{H}_i(f)] := \mathcal{D}_t[\mathcal{H}_i(f)]$. After all signa-

tures in the system have been processed, the working array is compressed into a single encryption element:

$$s_t = \prod_{j=1}^{|W_t|} w_{t,j}$$

where $w_{t,j}$ denotes the j-th element in working table t. The result for each table represents the bitmask corresponding to all classified signatures found in $\mathcal{F}$.

Administrator:Verify$(SK, c, \mathcal{C}, \mathcal{S})$**.** On input of a secret key, each element in $\mathcal{S}$ is decrypted and stored it in the set of plaintexts, $\mathcal{P}$.

For the input classified signature c, we check if the correct bits (based on permutations of $\mathcal{C}$) are set in the bitmask plaintexts. For each $t = \{1, ..., k\}$, let j_t be the index of $c \in PRP_t(\mathcal{C})$. Check if bit j_t is set in each plaintext $p_t \in \mathcal{P}$. Return 1 if all the bits are set correctly, otherwise return 0.

3.2 System Properties

A private signature searching scheme must have the following properties:

- **Correctness.** Verify is a probabilistic function such that for robustness parameter k:
  $\forall c \in \mathcal{C}, Pr[Verify(c, P) = 1 | c \in \mathcal{F}] \geq 1 - neg(k)$
- **Privacy.** Informally, an adversary should not learn the signatures being searched or even when they have been located.

4 ANALYSIS AND DISCUSSION

Choice of m. In this section, we quickly discuss our motivation for the two initial tests in the $Setup(\cdot,\cdot)$ algorithm: $|\mathcal{C}| \leq m$ and $m < \sqrt{|\mathcal{F}|}$. The first test simply to ensure that we have complete coverage in the mapping from signatures in $\mathcal{C}$ to bits in $\mathcal{Q}$. Without this one-to-one mapping, we cannot make any statements about the correctness of the server.

The second test, is to ensure that we can quickly find a valid hash index (one that produces no collisions) for $\mathcal{H}(\cdot)$. A well known probability result, known as the birthday paradox, tells us that given n bins and $\sqrt{n}$ balls the probability of having a single collision is $\frac{1}{2}$. By requiring $m < \sqrt{|\mathcal{F}|}$, the probability of collision will always be less than $\frac{1}{2}$. Thus, the probability of finding a valid hash index in k separate trials is greater than $1 - (\frac{1}{2})^k$.

Communication Complexity. The communication complexity of our scheme is asymptotically identical to the PIR schemes discussed earlier. However, we argue that administrator initialization is a one-time setup cost that can be amortized over several executions. This makes our approach preferable in situations where frequent scanning is expected.

4.1 Security Analysis

Adversarial Model. We assume the *honest-but-curious* adversarial model. In this model, servers execute the $Scan(\cdot,\cdot)$ algorithm honestly and do not intentionally or maliciously tamper with any output. They may, however, observe or record any intermediary algorithm state in an attempt to learn any final output behavior. We argue that this is reasonable since, within our context, the administrator is likely to have some form of authority over the servers it queries.

We argue that a stronger adversarial model does not make sense in our context. A malicious adversary, for instance, would either refuse to execute the $Scan(\cdot,\cdot)$ algorithm or skip over files during the scanning operation. It could also simply replace the final output with encryptions of random elements (which is possible given the Paillier public key).

Correctness and Privacy Proofs. Proof details have been omitted due to space constraints. However, we comment that the correctness proof follows from the Ostrovsky-Skeith proof and correctly verifies classified signatures with high probability. The privacy proof is simply a reduction to the semantic security property of the Paillier cryptosystem.

5 IMPLEMENTATION

We implemented our scheme in C++ and used multiple open source libraries. The Paillier cryptosystem implementation was based on the GNU Multiple Precision Arithmetic Library and used the OpenSSL library SHA-1 implementation for cryptographic hashing. SHA-1 was "keyed" by pre-pending the key to any data being hashed.

One optimization technique used was to perform multi-threaded table initialization for the data and working tables. Since both tables are initialized with $E(0)'s$, each thread perform a separate encryption operation. All encryptions were done using a 1024-bit Paillier public key (resulting in 2048-bit ciphertexts).

All tests were performed on a 64-bit Intel Core i7 960 / 3.2 Ghz CPU (4 CPU cores) running the Ubuntu 10.10 Linux distribution. We fixed the system robustness parameter (somewhat arbitrarily) at $k = 5$ and varied the number of files stored on the server, $|\mathcal{F}|$, to be scanned. Both the administrator and server al-

gorithms were benchmarked to gain an understanding of practical performance issues.

5.1 Administrator Benchmarks

Table 1: Administrator Benchmarks.

Setup$(\cdot,\cdot)$ $[k=5]$					
Experimental			Extrapolated		
$\lvert\mathcal{F}\rvert$	Init (min)	Storage (MB)	$\lvert\mathcal{F}\rvert$	Init (min)	Storage (MB)
10K	2.22	51	100K	22.34	512
20K	4.47	102	150K	33.51	768
30K	6.70	153	200K	44.68	1024
40K	8.97	204	250K	55.85	1280
50K	11.20	256	300K	67.02	1536

The initial results of the administrator benchmarks, reported in Table 1, indicate that the proposed scheme is a reasonable and practical approach for querying servers with both small and large datasets, $|\mathcal{F}|$. However, as the number of files increases, the administrator must decide when the data tables are too large to distribute. This will likely depend on whether the tables are transferred via the network or storage devices, e.g., USB memory stick.

The most expensive administrator operation is the data and working table initialization. However, because Paillier is a public key system, it is possible to shift some of this burden to the server. In particular, servers can compute their own working tables independently for each scan operation and reduce the required communication overhead by half.

Extrapolating to Large File Sets. Regardless of server file count, we average 1492 Paillier encryptions per second or $6.7\text{x}10^{-4}$ seconds per encryption. We extrapolate the expected initialization processing times and storage costs for larger file counts (right column of Table 1).

5.2 Server Benchmarks

For sever benchmarks, we scanned two large local system directories whose size closely approximated the table size generated by the administrator. The results, recorded in Table 2, records the time it took to (1) perform hashes of all files in the directory, and (2) perform all multiplications required to compress each working table into a single encryption element.

Table 2: Server Benchmarks.

Scan$(\cdot,\cdot)$ $[k=5]$			
Table Size	$\lvert\mathcal{F}\rvert$	Local Directory	Running Time (sec)
20K	18279	/usr/lib/	59.31
40K	39197	/usr/src/	539.33

Our results indicate that both operations can be performed efficiently. In general, the time taken to performing all hashing operations drastically exceeds the time taken to perform all multiplications. This is especially true when the files being hashed are large and require several hard disk fetches.

Note that because file size variability, extrapolating these results to larger data sets is not insightful. Total execution time is more accurately measured as a function of file size than as number of scanned files.

6 FUTURE WORK AND CONCLUSIONS

The scheme proposed represents the first steps towards an efficient and scalable solution for private searching of sensitive file signatures on untrusted servers. However, there are many potential areas for improvements and future work:

One possible direction is to consider reducing communication overhead in large networked environments. In our scheme, communication overhead grows linearly with the robustness parameter k. However, scanning all servers in a network simultaneously may overwhelm the administrator. It would be preferable to reduce overhead by supporting an in-network aggregation of all responses.

Another potential direction is to investigate techniques supporting alternate query forms, e.g., OR, AND, CNF. This would allow administrators to perform finer granularity queries for a specific situation.

In conclusion, we proposed a novel construction for private searching of sensitive file signatures, discussed implementation results, and show that our approach is efficient for administrators and servers.

REFERENCES

Bethencourt, J., Song, D., and Waters, B. (2009). New techniques for private stream searching. *ACM Trans. Inf. Syst. Secur.*, 12:16:1–16:32.

Freedman, M. J., Nissim, K., and Pinkas, B. (2004). Efficient private matching and set intersection. pages 1–19. Springer-Verlag.

Ostrovsky, R. and Skeith, III, W. E. (2007). Private searching on streaming data. *J. Cryptol.*, 20:397–430.

Paillier, P. (1999). Public-key cryptosystems based on composite degree residuosity classes. In *Proceedings of the 17th international conference on Theory and application of cryptographic techniques*, EUROCRYPT'99, pages 223–238, Berlin, Heidelberg. Springer-Verlag.