A Privacy-Preserving Infrastructure to Monitor Encrypted DNS Logs

Adam Oumar Abdel-rahman, Olivier Levillain, and Eric Totel

Samovar, Télécom SudParis, Institut Polytechnique de Paris, 91120 Palaiseau, France
adam_oumar.abdel_rahman@telecom-sudparis.eu,
olivier.levillain@telecom-sudparis.eu, eric.totel@telecom-sudparis.eu

Abstract. In the realm of cybersecurity, logging system and application activity is a crucial technique to detect and understand cyberattacks by identifying Indicators of Compromise (IoCs). Since these logs can take vast amounts of disk space, it can be tempting to delegate their storage to an external service provider. This requires to encrypt the data, so the service provider does not have access to possibly sensitive information. However, this usually makes it impossible to search for relevant information in the encrypted log. To address this predicament, this paper delves into the realm of modern cryptographic tools to reconcile the dual objectives of protecting log data from prying eyes while enabling controlled processing. We propose a comprehensive framework that contextualizes log data and presents several mechanisms to solve the outsourcing problem, allowing searchable encryption, and we apply our approach to DNS logs. Our contributions include the introduction of two novel schemes, namely symmetric and asymmetric, which facilitate efficient and secure retrieval of intrusion detection-related information from encrypted outsourced storage. Furthermore, we conduct extensive experiments on a test bed to evaluate and compare the effectiveness of the different solutions, providing valuable insights into the practical implementation of our proposed infrastructure for monitoring encrypted logs.

Keywords: Forensics · Indicators of Compromise · Searchable Encryption.

1 Introduction and Motivation

In the recent years, the amount of encrypted data (either in transit or at rest) has increased drastically. This is *a good thing* from the point of view of the privacy since it significantly reduces the exposition of personal data to cyberattacks. This, however, poses new challenges for information system administrators whose monitoring solutions have become obsolete in the face of user-generated traffic. Should we keep the data relevant to intrusion detection systems (IDSes) in cleartext at the cost of the confidentiality of communications by exposing sensitive information to an external third party? Or should we accept that our detection tools have become blind?

In the realm of intrusion detection, the necessity to maintain system and application logs often results in the accumulation of extensive data volumes, a scenario that lends itself to potential outsourcing. Moreover, the sheer magnitude of data generated through DNS logs presents a significant predicament for organizations, prompting meticulous deliberation regarding storage solutions. A standard DNS log entry, contingent upon its complexity, consumes between 100 and 1000 bytes of storage. Each operational day, a single machine generates between 1000 and 10,000 DNS requests, each contributing to the accumulation of log data. For an organization comprising 50 machines, operating on the premise of 200 working days annually, the yearly DNS log data volume can oscillate from 1 gigabyte (10^9 bytes) to a staggering 100 gigabytes (10^{11} bytes). This underscores the substantial storage requisites that organizations confront in effectively managing these logs. Notably, this circumstance further accentuates the imperative to reconcile ostensibly conflicting objectives: the imperative to encrypt the data to safeguard it from the service provider handling the logs, while concurrently permitting a certain level of processing to enable the retrieval of information requisite for attack detection. In practical terms, this mandates that the service provider possesses the capability to process the encrypted logs effectively.

DNS is usually central to detect intrusions in retrospect. Indeed, in many cases, malware need to call their Command & Control (or C2) server to get instructions on the action to execute. This communication usually triggers DNS requests at some point. Information about these requests (the requested domain name or the returned IP addresses), which are called Indicators of Compromise (IoCs), can be shared to detect attacks (or even block them sometimes).

By leveraging advanced cryptographic tools, we propose in this paper to encrypt the logs while allowing a partial, controlled and delegatable search capability to the service provider. We thus describe and implement solutions to encrypt DNS logs and allow for specific requests, e.g. to find the DNS queries sent for a given domain name, on them later.

In this paper, our contributions are threefold. First, we describe the DNS context in details, including the logs to store, the relevant requests and the threat model we consider. Then we study and implement several mechanisms, using either symmetric and asymmetric cryptography, to allow for efficient searchable encryption in our precise context. Finally, we design and run a test bed to experiment on the different solutions and compare them.

2 DNS Use Case

2.1 Description

Every usual operation (such as accessing a website, checking an e-mail, or logging into a user account via an application) starts with a name resolution. This task is handled by the DNS resolver. Since cyberattacks usually trigger DNS requests, one of the ways to deal with these incidents is to identify traces of Indicators of Compromise in DNS logs (such as domain names or IP addresses corresponding to a malicious server).

To illustrate our use case shown on Figure 1, we consider a company with one or several internal DNS resolvers and employees using these resolvers. The company wants to delegate storage and monitoring of its DNS logs to an external service provider. Since DNS logs contain sensitive information, they require encryption to guarantee confidentiality. Additional guarantees such as search capabilities are required. As the goal is to reconcile privacy-preserving storage with the ability to search for IoCs in an efficient and outsourced manner, we need a suitable cryptographic scheme. We describe such schemes in Sec. 3.

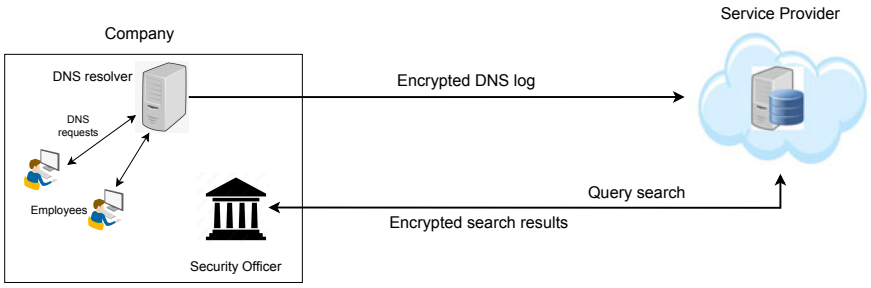

Fig. 1. Platform Illustration

In our reference scenario, when an employee sends a query for a domain name, the DNS resolver responds and generates the corresponding DNS log entry. This entry is then encrypted and transmitted to a designated service provider in charge of storing and monitoring the encrypted logs. While we assume the service provider to be honest in performing their assigned tasks, they may possess a level of curiosity and attempt to extract sensitive information from the logs. To facilitate Indicators of Compromise (IoC) detection, the company's security officer, responsible for IT security, has the authority to grant search capabilities for specific IoCs to the service provider. This enables the service provider to conduct searches over the encrypted logs. It is important to define searchable attributes within the logs to facilitate these search operations accurately.

Each log entry associated to a DNS query, that we call a DNS log, is a record containing the following information:

- the timestamp of the query;
- the DNS resolver identifier (in case the company has several resolvers);
- the IP address of the employee who makes the request;
- the domain name queried;
- the type of the query (`A`, `AAAA`, `MX` etc.);
- the response code (which describes the result/status of the request, e.g. `NOERROR`, `NXDOMAIN`);
- the list (possibly empty) of IP addresses returned to the employee.

To allow for future IoC lookup, each DNS log entry is linked with both the requested domain name and all the IP addresses provided in response to that specific request. We refer to these attributes, whether it's the domain name or the IP addresses, as *keywords*. On average, a log entry is associated with approximately 5 keywords. When a *keyword* is later used during a query search, because it is supposedly related to malicious activity, we talk about an *IoC*.

As described above, we need to share DNS logs that contain sensitive information with a service provider that is supposed to be honest but curious. We therefore need a threat model to define the required security properties.

2.2 Security Properties

The adopted threat model assumes the service provider honestly performs all their tasks, but tries to obtain confidential information by analyzing the received encrypted logs, search queries, and matching results to obtain information about the content of the logs. Moreover, we assume the existence of external threats.

Log Unforgeability Unauthorized party (including the service provider) should not be able to forge a valid encrypted log. This means that only DNS requests sent to a legitimate resolver should be present in the logs.

Predicate Privacy In order to analyze the stored encrypted logs, the security officer must provide the search capability to the service provider which we refer to as a *trapdoor*. A *trapdoor* is generated using a secret key owned by the security officer and an IoC. We expect the service provider to extract all encrypted logs relevant to this IoC. In this scenario the IoC is considered sensitive, thus a trapdoor should not reveal any information on the encapsulated IoC. This requirement is known as *Predicate Privacy* [10].

Correlation Privacy Finally the search results of the analysis should not reveal any information to the service provider except the number of matching entries. In particular, the knowledge of a trapdoor and an encrypted entry matching the IoC represented by the trapdoor should not reveal any information about the plaintext log associated with this entry.

3 Searchable Encryption

To reconcile data encryption and the ability to search for IoCs in the logs, a trivial but naive solution is to download the complete logs from the server, decrypt them, and search for IoCs in the plaintext data. Obviously, this is an inefficient and costly approach due to the expense of downloading large amounts of data on the client side. Thus, it is desirable to support search functionality at the server level without having to decrypt all the data. To address this issue, *Searchable Encryption* (SE) has been proposed as a mechanism to encrypt data while supporting keyword search over the encrypted data, without requiring access to

plaintext. Searchable encryption was first introduced and defined formally by Song et al. [12].

As described by Bösch et al. [7], SE can be used either for data outsourcing or sharing. In the first case, outsourcing, also known as *Symmetric Searchable Encryption* (SSE), the secret key holder both produces ciphertexts and search queries using symmetric primitives. The first SSE was proposed by Song et al. [12]. In the second case, which corresponds to data sharing, we use a public key setting called *Asymmetric Searchable Encryption* (ASE). Such a scheme allows multiple users (in possession of the public key) to generate searchable ciphertexts, and only the private key holder controls the ability to perform encrypted search. The first ASE is due to Boneh et al. [4] who proposed a *Public key Encryption with Keyword Search* (PEKS) scheme.

Informally, in searchable encryption schemes, there is a message and a set of associated keywords. During encryption, the user encrypts the message with a symmetric key encryption, then, generates *searchable encrypted indexes* from these keywords, which we call *tokens*. In the SSE, a *token* may be generated by using a one-way function as a hash function or pseudo-random function (PRF) [8,6], while in the ASE, a *token* may be a ciphertext corresponding to a keyword. Let M be a message with associated keywords $kw_1, \ldots, kw_n$. We denote the searchable ciphertext as $\{Enc_K(M), TK_1, \ldots, TK_n\}$ where each TK_i is a *token* corresponding to the keyword kw_i.

In our formalism, an SE scheme is defined by five algorithms: The `Setup` algorithm generates the cryptographic setting. It takes a security level and generates the public and private parameters. The `Encryption` algorithm takes a message M, a set of keywords $kw_1, \ldots, kw_n$ associated to M and the public parameters (for an ASE scheme) or the secret key (for an SSE scheme). Then this algorithm produces a searchable ciphertext output. The `TrapdoorGen` algorithm takes a keyword kw and private parameters and then produces a *trapdoor* $T(kw)$ for that keyword. The trapdoor $T(kw)$ is a search capability who authorizes the service provider to only process encrypted entries relevant to the keyword kw. When a trapdoor $T(kw)$ is sent to the service provider, it runs the `Search` algorithm to extract all encrypted entries that match the keyword kw. Finally, the `FinalDecryption` algorithm is required to decrypt the plaintext logs from the search results. The complete cinematics is represented in Figure 2 in Section 4.2.

SE schemes are mostly used to secure sensitive data outsourcing or sharing. It is thus important to consider the privacy and security requirements, which include the confidentiality protection of the data, the privacy of the search queries, the prevention of unauthorized access either to the data or search results, and the guarantee that the search process does not reveal any information about the data (Correlation Privacy) or the search queries (Predicate Privacy) [4,10].

Additionally, the SE scheme should be efficient and scalable to support practical applications.

4 Application of Searchable Encryption to DNS Logs

4.1 Studied Schemes

In this section, we present three cryptographic schemes to create encrypted and searchable entries. We first present the symmetric scheme proposed by Waters et al. [13] and a new symmetric solution based on PRFs. The third scheme is an ASE based on Identity-Based Encryption.

All the schemes rely on symmetric primitives, namely an authenticated encryption scheme E_k and a pseudo-random function PRF.

WBDS–SSE scheme We now present an SSE proposed by Waters et al. in [13].

`Setup`. The security officer generates a random secret K_R and shares it with the DNS resolver R.

`Encryption`. We assume the security officer and the resolver R share the K_R secret to encrypt log entries. Let `record` be a DNS log along with keywords $kw_1, \ldots, kw_n$. Let `flag` be a constant bitstring of length l[1]. The resolver executes the following steps

1. Choose a random symmetric key K and compute $E_K(\texttt{record})$.
2. Choose a random bitstring r of some fixed length[2].
3. For each keyword kw_i, compute

$$a_i = \texttt{PRF}_{K_R}(kw_i), \quad b_i = \texttt{PRF}_{a_i}(r), \quad c_i = b_i \oplus (\texttt{flag}|K).$$

4. Return the encrypted DNS log $\{E_K(\texttt{record}), r, c_1, \ldots, c_n\}$.

The output will be sent to the service provider.

`TrapdoorGen`. When the company wants to look for an IoC, the security officer generates the search capability by evaluating a PRF keyed by the secret key K_R on the IoC denoted by $K_{IoC} := \texttt{PRF}_{K_R}(IoC)$, and sends it to the service provider.

`Search`. When the service provider receives K_{IoC}, it executes the following algorithm:

1. For each entry $\{E_K(\text{record}), r, c_1, \ldots, c_n\}$
   (a) Compute $p = \texttt{PRF}_{K_{IoC}}(r)$.
   (b) For each c_i in the entry, compute $p \oplus c_i$. If the first l bits of the result matches **flag**, extract K as the remainder of the result; otherwise, the computation is disregarded. If none of the results starts with `flag`, move to the next query (the query is not a keyword match).

[1] The flag can have a length significantly less than that of an encryption key K [13].

[2] The length of r does not affect the security properties of the scheme with regards to the expected security properties since it is the input to a PRF. The impact of the size of r on other properties is discussed in Sec. 5.3.

   (c) If one of the results matches, use the computed K to decrypt $E_K(\text{record})$ to obtain the record in clear text form.
2. Send the (possibly empty) list of gathered records to the security officer.

Note that this steps directly produces the plaintext entries, letting the service provider have access to them. It also makes the `FinalDecryption` step trivial.

An indexable, PRF-based SSE scheme We propose a different approach from the previous scheme. The idea is to deterministically derive tokens from the keywords and a truncated version of the timestamp[3]. A trapdoor for our scheme is simply the corresponding token, which allows to efficiently retrieve the relevant lines, e.g. by using database indexes. The rest of the record (and the search results) are encrypted to preserve the confidentiality of logs.

The `Setup` algorithm is the same as the WBDS–SSE scheme and suppose that the security officer share the secret key K_R with the resolver R.

`Encryption`. Let `record` be a DNS log along with keywords $kw_1, \ldots, kw_n$ and a timestamp TS. We run the following steps to generate the encrypted entry.

1. For each keyword kw_i, compute the associated token TK_i by evaluating the PRF $TK_i = \texttt{PRF}_{K_R}(kw_i||\overline{TS})$ where $\overline{TS}$ is the truncated version of TS.
2. Generate the symmetric key $K = \texttt{PRF}_{K_R}(TK_1||\ldots||TK_n)$.
3. Compute the encryption $E_K(\text{record})$.
4. Return the encrypted DNS log $\{E_K(\text{record}), TK_1, \ldots, TK_n\}$.

The resolver sends $\{E_K(\text{record}), TK_1, \ldots, TK_n\}$ to the service provider who stores it in its database.

`TrapdoorGen`. In case one wants to recover all the queries associated with an IoC within some time frame $[T_A, T_B]$, the security officer executes the following steps to generate the trapdoor corresponding.

1. Generate $K_{IoC} = \text{PRF}_{K_R}(IoC)$.
2. Find the minimal set of truncated timestamps $\overline{TS_1}, \ldots, \overline{TS_m}$ covering the time frame $[T_A, T_B]$.
3. For each t_i generate the corresponding token $tk_i = \text{PRF}_{K_R}(IoC||\overline{t_i})$.
4. Return the trapdoor $T(IoC) = \{tk_1, \ldots, tk_m\}$.

`Search`. Suppose that the service provider receives a query search (a trapdoor $T(IoC) := \{tk_1, \ldots, tk_m\}$) from the security officer. The service provider works as follows to retrieve all queries matching the encrypted IoC. For each entry $\{E_K(\text{record}), TK_1, \ldots, TK_n\}$ in the encrypted database, check the intersection of token sets $\{TK_1, \ldots, TK_n\} \cap \{tk_1, \ldots, tk_m\}$. If this intersection is non-empty set, i.e., there exist $i \in \{1, \ldots, n\}$ and $j \in \{1, \ldots, m\}$ such that $TK_i = tk_j$, the

[3] A typical value for the truncation windows would be 1 hour or 1 day. Small windows will lead to the multiplication of trapdoors, whereas big windows will make token collision more probable (see Sec. 5.3 for a discussion on the matter).

entry match, then add the entire entry (encrypted record with all its tokens) to the search results. Finally, send the search results to the security officer.

`FinalDecryption`. When the security officer receives the search results, it can derive, for each entry $\{E_K(\text{record}), TK_1, \ldots, TK_n\}$ the decryption key $K = \texttt{PRF}_{K_R}(TK_1 || \ldots || TK_n)$, and finally decrypt $E_K(\texttt{record})$ to recover the plaintext record.

Asymmetric Searchable Encryption (ASE) using IBE We propose a solution based on an asymmetric approach. We first present the Identity-Based Encryption (IBE) scheme of Boneh and Franklin [5], and then the ASE scheme built on it.

Identity-Based Encryption IBE is a public key scheme where any arbitrary string is a valid public key. The corresponding private key is generated by a trusted third-party from the public key. The canonical example of IBE is to use one's identity (e.g. an email address) as the public key, hence the name of the scheme.

In our use case, we use the keywords associated to the DNS logs as public keys (or identities); the third party generating private keys is the security officer.

Boneh and Franklin proposed the FullIdent IBE protocol [5], which is proven to be IND-CCA under *BDH* assumption[4]. This scheme, on which we build the ASE scheme, defines the following operations.

`IBE.Setup`. Choose a large prime number q, two groups $\mathbb{G}_1$ and $\mathbb{G}_2$ of order q and choose an admissible bilinear map $\mathrm{e} : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$. Choose a random generator P of $\mathbb{G}_1$, a random master key $s \in \mathbb{Z}_q^*$. We also need four hash functions $H_1 : \{0,1\}^* \to \mathbb{G}_2^*$, $H_2 : \mathbb{G}_T \to \{0,1\}^n$, $H_3 : \{0,1\}^n \times \{0,1\}^n \to \mathbb{Z}_q^*$ and $H_4 : \{0,1\}^n \to \{0,1\}^n$. Finally the authority publish the public parameters $\texttt{params} = \langle q, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, \mathrm{e}, n, P, P_{pub}, H_1, H_2, H_3, H_4 \rangle$ where $P_{pub} = sP \in \mathbb{G}_1$ and keep secret the master key s.

`IBE.Extract`. Given an $ID \in \{0,1\}^*$ as a public key, this algorithm computes the corresponding private key $SK_{ID} = sH_1(ID) \in \mathbb{G}_2$ where s is the master key.

`IBE.Encrypt`. To encrypt a message $M \in \{0,1\}^n$ under the public key ID :

1. Compute $Q_{ID} = H_1(ID) \in \mathbb{G}_2$.
2. Choose a random bitstring $\sigma \in \{0,1\}^n$ and set $r = H_3(\sigma, M) \in \mathbb{Z}_q^*$.
3. The ciphertext is $C = \langle rP, \sigma \oplus H_2(e(P_{pub}, Q_{ID})^r), M \oplus H_4(\sigma) \rangle$.

`IBE.Decrypt`. Let $C = \langle U, V, W \rangle$ a ciphertext encrypted using the public key ID. If $U \notin \mathbb{G}_1$, reject the ciphertext. To decrypt C using the private key SK_{ID}:

1. Compute $\sigma = V \oplus H_2(e(U, SK_{ID}))$ and $M = W \oplus H_4(\sigma)$.
2. Set $r = H_3(\sigma, M)$. Test that $U = rP$. If not, reject the ciphertext.

[4] Assuming the *Bilinear Diffie-Hellman Problem (BDH)* is hard (i.e. all polynomial time algorithms have a negligible advantage in solving BDH), FullIdent is semantically secure against an adaptive chosen ciphertext attack (IND-CCA) [5].

3. Return M as the decryption of C.

The decryption function is a kind of try-decrypt, which is particularly interesting for our use case. Let $C = \texttt{IBE.Encrypt}(\texttt{params}, ID, M)$ be a ciphertext and SK_{ID^*} a private key (corresponding to the public key ID^*) generated by the `IBE.Extract` algorithm. The decryption of C success if and only if C is well formed and $ID^* = ID$.

ASE scheme The IBE-based ASE scheme relies on the following algorithms.

`Setup`. First, run the `IBE.Setup` algorithm to generate public parameters `params` and the master key s. Then, draw a random secret key K_R which is shared with the resolver. Public parameters are shared with the different actors.

`Encryption`. Let `record` be a DNS log along with keywords $kw_1, \ldots, kw_m$, the DNS resolver executes the following steps to generate the encrypted and searchable entry:

1. Choose a random secret key K.
2. For each keyword kw_i, compute $C_i = \texttt{IBE.Encrypt}(K, kw_i)$ i.e. we encrypt K using the keywords kw_i as public keys.
3. Compute the encryption $E_{K'}(\texttt{record})$ where $K' = \texttt{PRF}_{K_R}(K)$.
4. Return the encrypted DNS log $\{E_{K'}(\texttt{record}), C_1, \ldots, C_m\}$.

The resolver sends the result of this algorithm with a timestamp `TS` which can be stored by the service provider in form $\{E_{K'}(\texttt{record}), \texttt{TS}, C_1, \ldots, C_m\}$.

`TrapdoorGen`. When the security officer wants to search for an IoC, it runs `IBE.Extract` algorithm to generate SK_{IoC} the private key corresponding to the public key IoC, which is trapdoor. It sends the trapdoor with the time frame of interest $\{SK_{IoC}, [T_A, T_B]\}$ to the service provider which processes encrypted logs received in the $[T_A, T_B]$ period.

`Search`. Let SK_{IoC} a trapdoor and $[T_A, T_B]$ the time frame sent to the service provider. For each encrypted record $\{E_{K'}(\texttt{record}), TS, C_1, \ldots, C_m\}$ within the given time frame, the service provider executes the following steps.

1. For each C_i, it tries to decrypt C_i with `IBE.Decrypt` using the private key SK_{IoC}. If the decryption succeeds, C_i corresponds to IoC and the secret $K := \texttt{IBE.Decrypt}(C_i, SK_{IoC})$ is recovered. It thus adds $\{E_{K'}(\texttt{record}), K\}$ to the search results and moves to the next log entry.
2. If none of the C_i can be decrypted, the entry does not match the IoC and it moves to the next entry.

In the end, the service provider sends all the relevant search results to the security officer.

`FinalDecryption`. For each entry $\{E_{K'}(\texttt{record}), K\}$, the security officer can derive the decryption key $K' = \texttt{PRF}_{K_R}(K)$, and finally decrypt $E_{K'}(\texttt{record})$ to recover the plaintext record.

4.2 Platform Implementation

Our proposed infrastructure for monitoring DNS logs using searchable encryption is designed to provide secure and efficient monitoring of DNS logs. As shown in Figure 2, the infrastructure consists of several actors that work together to achieve this goal. We describe in detail in this section the main task of these actors.
Step 1–2 (initialization and sharing secret key) At the beginning, the security officer runs the `Setup` algorithm to initialize cryptographic materials for the considered scheme (SSE or ASE) and shares the secret key K_R with the DNS resolver. The public parameters, including cryptographic primitives and public key (in case of ASE) are shared with the resolver and the service provider.

Step 3 (processing log) At the core of the infrastructure is a DNS resolver, which generates DNS logs. For each DNS query, the resolver extracts the keywords from the DNS log, constructs the `record` and encrypts it using `Encryption` algorithm. The `Encryption` takes as input the `record` and a set of keywords and produces a searchable ciphertext as described in section 3. The resolver sends the `Encryption` output (including the tokens), to the service provider who stores them in its database.

Step 4 (searching over encrypted logs) To perform a search query on the encrypted logs, the security officer generates the search capabilities for an IoC by running the `TrapdoorGen` algorithm and sends them to the service provider. When the service provider receives a query from the security officer, it runs the `Search` algorithm which takes as input a search capability. It retrieves all matching records from the encrypted logs stored in the database and sends results to the security officer.

Step 5 (decrypting search results) When the security officer receives the search results for its query search, it runs the `FinalDecryption` algorithm to decrypt them and recover the plaintext records.

Implementation details We implemented our test bed in C. Symmetric primitives come from OpenSSL. For the indexable PRF-based scheme, the database is SQLite. Finally, for the asymmetric primitives, we use the RELIC library, a modern, research-oriented, cryptographic meta-toolkit with emphasis on efficiency and flexibility [1]. The IBE scheme we use indeed requires an asymmetric pairing-friendly setting $(q, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, \mathrm{e}, P, P_{pub})$. RELIC implements several elliptic curves; it also implements hash functions on elliptic curves and asymmetric pairings.

For our implementation parameters, we use the BLS12-381 elliptic curve [3]. In terms of symmetric primitives, we have employed E = AES-256-GCM for encryption and PRF = HMAC-SHA-2 for PRF. The selection of the BLS12-381 curve for our IBE-based ASE scheme is underpinned by a comprehensive assessment that considers established standards and practical suitability. This particular curve has gained widespread recognition and adoption in prominent

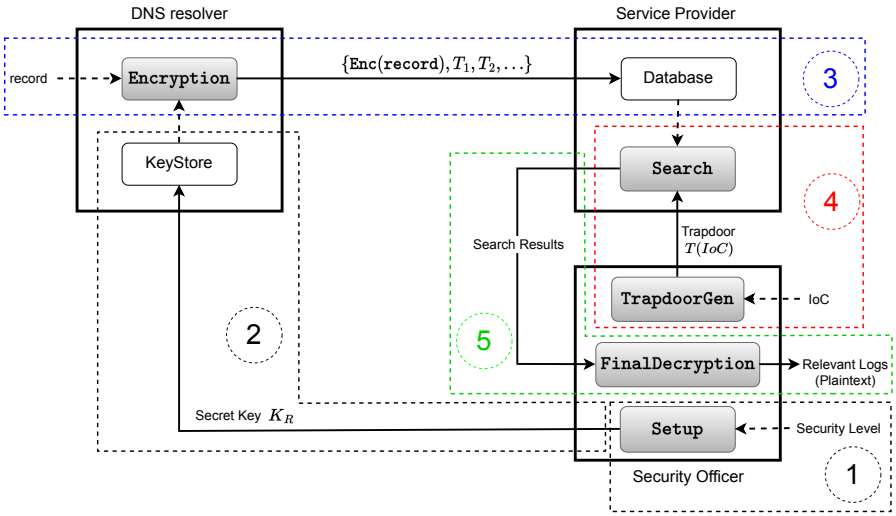

Fig. 2. Platform Architecture

blockchain projects such as Zcash and Ethereum 2.0, a testament to its resilience and its alignment with the stringent requirements of security-sensitive applications. Furthermore, our choice aligns with the recommendations outlined by the IETF [9], which emphasize the merits of the BLS12-381 curve. This alignment with industry best practices, particularly for achieving a 128-bit security level, underscores the curve's status as both *widely used* and *efficient*. These attributes collectively reinforce its credibility as a secure and pragmatic choice for our cryptographic setup.

5 Evaluation

5.1 Security

In the WBDS–SSE scheme, the log is encrypted using a random key, which means anyone can generate valid encryption. Only the generation of the tokens requires the knowledge of the secret key K_R. However, knowing a trapdoor is sufficient to forge a valid token (for the corresponding keyword), which means the *Log Unforgeability* property is not completely satisfied, especially w.r.t. the service provider which is supposed to eventually learn trapdoors. In this scheme, the trapdoor for a keyword results from a PRF keyed by the secret key K_R. The *Predicate Privacy* property is a direct result of the PRF properties.

Finally, by construction, the original scheme described by Waters et al. does not provide *Correlation Privacy*. A simple way to add this property would be to encrypt the record with a key shared only between the resolver and the security officer before running the scheme. Adding this encryption step would also guarantee *Log Unforgeability*.

In our PRF-based SSE scheme the tokens and the encryption key are calculated using a PRF keyed by a secrete key. This implies that an attacker who does not know this secret key cannot generate a valid entry, which guarantees the *Log Unforgeability* property. At the same time, the guarantee of *Predicate Privacy* and *Correlation Privacy* properties follows directly from the security properties of the PRF and from the secrecy of K_R.

For our IBE-based ASE, a token is the ciphertext returned by the encryption algorithm of the IBE, which is a public key encryption. That means an attacker can generate valid tokens. They can also create an arbitrary $E_{K'}(\texttt{record})$ element. Yet, they can not create a consistent ciphertext where K' is linked to the C_i, thanks to the PRF security properties. Thus, the *Log Unforgeability* property is guaranteed. The *Predicate Privacy* follows directly from the definition of the secret key SK_{ID} from the public key, via a one-way hash function. Finally, since the actual record is protected by the secret key K_R, the *Correlation Privacy* is guaranteed by the security properties of the authenticated encryption primitive.

5.2 Benchmarking

We evaluate the effectiveness and scalability of our proposed infrastructure using a realistic DNS log dataset [11]. We compare the performance of our infrastructure with a baseline approach that uses plaintext log data and a conventional search function. Since the plaintext and the PRF-based schemes are compatible with database indexes, we also provide figures for a variant thereof using a database back-end.

We evaluated the studied with 21 PCAP files (each files corresponds to 1-hour long captures form) of size 6.29 GB from [11]. Among these files, the DNS response packets of interest amounted to 3.45 GB. We preprocess these PCAP files to generate the corresponding record files (according to the records format described in section 2.1). These record files make a total of $14,816,004$ records and a size of 763 MB. For our experiments, we ran the whole cinematics (from `Setup` to `Search` and `FinalDecryption` with different IoCs) on those files. The experiments were run on a computer running Linux with a 32-core Intel Xeon Gold 5218R CPU, 2.10GHz and 64GB of RAM memory.

The results in Table 1 clearly show that the IBE cost are orders of magnitude bigger in terms of processing time. For the schemes which are compatible with database indexes, using an SQLite back-end leads to bigger storage requirements, and slows down significantly the encryption time, but it allows for drastic improvements in the search time. This is indeed expected since these schemes allow to retrieve the relevant lines using an index, without the need for testing each line, which can be a game changer when looking at a handful of records in huge logs.

5.3 Discussion About Token Collision

In the PRF-based SSE, the tokens are directly derived from keywords and a truncated timestamp, therefore, within a truncation interval, the token for a

	Encryption Time (μs/log)	**Size of Searchable Ciphertext (in bytes)**	**Search Time (s/IoC)**
Plaintext (baseline)	2.7	1.0 (102)	0.4
Plaintext + DB	130.0	2.4 (248)	< 0.01
WBDS–SSE [13]	22.4	2.3 (236)	2.2
PRF-based SSE	28.9	1.3 (129)	9.97
PRF-based SSE + DB	134.4	3.3 (343)	0.02
IBE-based ASE	5569.0	4.7 (480)	2189.28

Table 1. Comparison of different schemes using a public DNS data set [11]. Schemes with a "+ DB" annotation correspond to variants which allows for indexable search and where the data is stored in an SQLite back-end. The Encryption Time is per DNS log. The Size of Searchable Ciphertext is given relative to the plaintext baseline, which averages around 102 bytes. The Search Time corresponds to the processing time of one IoC in 705,524 encrypted lines logs (1-hour long capture).

given keyword is deterministic. Such leakage allow the service provider to derive statistics on the encrypted logs, including those unrelated to IoCs.

With WBDS–SSE, the collision are subtler. As discussed earlier, a collision on the masked `flag` (of length l) may arise by accident, with probability 2^{-l}. However, two logs pertaining to the same keyword can be encrypted with the same random r, which will lead to the same b_i. This will happen with probability $2^{-|r|/2}$. So, if $|r|$ is small and significantly less than l, collisions may reveal the presence of a common keyword between two records.

We believe this leakage is small and should not be problematic for the studied use cases. This might however require further analysis. In the meantime, a conservative measure would be to choose parameters reducing the collision probability (long enough random bitstring r for WBDS–SSE or small truncation windows for the PRF-based scheme).

5.4 Summary

	Log Unforgeability	Predicate Privacy	Correlation Privacy	Token Collisions	Search Efficiency
WBDS–SSE [13]	✗	✓	✗	if $\|r\| \ll l$	+
PRF-based SSE	✓	✓	✓	Within the truncation window	++
IBE-based ASE	✓	✓	✓	No	--

6 Related Work

In recent years, there has been a growing interest in the use of searchable encryption techniques for secure and efficient searching of encrypted data. One of the earliest works in this area was published in 2000. In this paper, Song et al. [12] proposed a symmetric key-based scheme. In their scheme, the message to

be encrypted is seen as a sequence of keywords. The main idea is to achieve, for each keyword, a special two-layered encryption construct. Thus, given a trapdoor, the server can strip the outer layer and check if the inner layer is in the correct form. This scheme works on text content and allow to filter ciphertexts containing a given keyword. Since the schemes we study take advantage of the underlying data structure (domain names, IP addresses, etc.), our results are more efficient, both in terms of processing time and encrypted data volume.

Boneh et al. [4] introduced the idea of using public key encryption to enable search over encrypted data. They provide a precise definition of what they call *Public key Encryption with Keyword Search (PEKS)* along with several constructions that are provably secure in their model under suitable cryptographic assumptions. Our work proposes a suitable construction using a hybrid symmetric scheme for monitoring DNS logs securely.

Building on these foundational works, several research efforts have explored the use of searchable encryption for secure logging. Waters et al. [13] proposed an encrypted and searchable audit log. They proposed two approaches: an SSE, which we introduced in Section 4.1 and an ASE similar to the construction presented in this paper. Our work proposes a new symmetric scheme, which is compatible with database indexes. We validate the performance of this scheme with regards to schemes proposed by Waters et al. [13].

More recently, Araújo and Pinto [2] proposed a system for secure remote storage of logs while maintaining its confidentiality and server-side search capabilities. This work uses symmetric and asymmetric encryption algorithms to enable secure and efficient search over encrypted log data. The system provides very expressive queries, however, their experiments were only run on small datasets (they target logs from web servers, with 30,000 lines), which makes it difficult to assess the scalability in use cases as ours.

7 Conclusion

In this work, we design an architecture for monitor encrypted DNS logs while ensuring the privacy preserving of the DNS logs. To do so, we leverage the use of searchable encryption. We have proposed two solutions which are symmetric and asymmetric approaches. We also studied the SSE scheme proposed by Waters et al. [13]. We set up a platform with the different actors and implemented the studied schemes.

Our symmetric approach allows for efficient and secure searching of encrypted DNS logs, while preserving the privacy of sensitive information. We evaluate our approach using real-world DNS log data and demonstrate its effectiveness in terms of performance and security. By using our infrastructure, companies could outsource and monitor their DNS logs while ensuring the privacy and security of the logged data.

Beyond DNS logs, we believe our work can be extended as future work to more complex and more sensitive data, such as system logs or mail-related logs. However, this extension would raise interesting questions to accommodate the

constraints related to these logs, e.g. mail logs contain both structured and non-structured (textual) data which might be hard to reconcile.

Acknowledgment

This work was supported by the French ANR Project ANR-19-CE39-0011 PRESTO.

References

1. D. F. Aranha, C. P. L. Gouvêa, T. Markmann, R. S. Wahby, and K. Liao. RELIC is an Efficient LIbrary for Cryptography. https://github.com/relic-toolkit/relic.
2. Rui Araújo and António Pinto. Secure Remote Storage of Logs with Search Capabilities. *Journal of Cybersecurity and Privacy*, 1(2):340–364, 2021.
3. Paulo SLM Barreto, Ben Lynn, and Michael Scott. Constructing elliptic curves with prescribed embedding degrees. In *Security in Communication Networks: Third International Conference, SCN 2002 Amalfi, Italy, September 11–13, 2002 Revised Papers 3*, pages 257–267. Springer, 2003.
4. Dan Boneh, Giovanni Di Crescenzo, Rafail Ostrovsky, and Giuseppe Persiano. Public Key Encryption with Keyword Search. In Christian Cachin and Jan L. Camenisch, editors, *Advances in Cryptology - EUROCRYPT 2004*, pages 506–522, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.
5. Dan Boneh and Matthew K. Franklin. Identity-Based Encryption from the Weil Pairing. In Joe Kilian, editor, *Advances in Cryptology - CRYPTO 2001, 21st Annual International Cryptology Conference, Santa Barbara, California, USA, August 19-23, 2001, Proceedings*, volume 2139 of *Lecture Notes in Computer Science*, pages 213–229. Springer, 2001.
6. Dan Boneh and Brent Waters. Constrained pseudorandom functions and their applications. In Kazue Sako and Palash Sarkar, editors, *Advances in Cryptology - ASIACRYPT 2013 - 19th International Conference on the Theory and Application of Cryptology and Information Security, Bengaluru, India, December 1-5, 2013, Proceedings, Part II*, volume 8270 of *Lecture Notes in Computer Science*, pages 280–300. Springer, 2013.
7. Christoph Bösch, Pieter Hartel, Willem Jonker, and Andreas Peter. A survey of provably secure searchable encryption. *ACM Computing Surveys (CSUR)*, 47(2):1–51, 2014.
8. Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions. *Journal of the ACM (JACM)*, 33(4):792–807, 1986.
9. Y Sakemi, T Kobayashi, T Saito, and R Wahby. Pairing-friendly curves, 2021. *IETF draft*.
10. Emily Shen, Elaine Shi, and Brent Waters. Predicate privacy in encryption systems. In *Theory of Cryptography: 6th Theory of Cryptography Conference, TCC 2009, San Francisco, CA, USA, March 15-17, 2009. Proceedings 6*, pages 457–473. Springer, 2009.
11. Manmeet Singh, Maninder Singh, and Sanmeet Kaur. TI-2016 DNS dataset, 2019.
12. Dawn Xiaoding Song, D. Wagner, and A. Perrig. Practical Techniques for Searches on Encrypted Data. In *Proceeding 2000 IEEE Symposium on Security and Privacy. S P 2000*, pages 44–55, 2000.
13. Brent Waters, Dirk Balfanz, Glenn Durfee, and Diana K. Smetters. Building an Encrypted and Searchable Audit Log. In *NDSS*, 2004.