Lectures 4+5: The (In)Security of Encrypted Search

Contents

1 Overview

In the first lecture, we went over—at a high level—the different ways to search on encrypted data. We also pointed out that each solution achieved a different trade-off between efficiency, security and expressiveness. In the second lecture, we argued that the security of a cryptosystem should be analyzed in the provable/reductionist security paradigm. While this paradigm has its limitations, it is the best way we have to reason about the security of cryptosystems and to debug our algorithms.

Here, we apply the reductionist security paradigm to the problem of encrypted search. We give syntax and security definitions that capture what encrypted search solutions look like and the security properties they should achieve. These definitions will make our discussions in future lectures more precise and allow us to compare and contrast the algorithms we will study.

The meaning of search. Before we can properly formalize encrypted search we have to pin down what we mean by search as it means different things depending on the context. In search algorithms, we consider two complexity regimes, linear and sub-linear, and two algorithmic paradigms, structured and unstructured. In the linear regime we allow algorithms that run in $O(n)$ time, where n is the length of the data being searched. In the sub-linear regime we only allow algorithms that run in $o(n)$ time. In the structured paradigm, we allow a $O(n)$ setup phase to pre-process the data and a query phase which is typically sub-linear. In the unstructured paradigm, we do not allow any pre-processing.

For the most part, we will ignore linear search solutions. Structured linear solutions are of no interest and unstructured linear solutions essentially correspond to sequential scan which is prohibitive for the kinds of datasets we are interested in (on the order of GBs or larger). We note that structured sub-linear and unstructured sub-linear algorithms roughly correspond to the fields of data structures and sub-linear algorithms [1], respectively. The former achieve sub-linearity at the cost of additional storage (needed for the structure) whereas the latter achieve sub-linearity at the cost of errors.

For search applications (e.g., search engines or databases) the structured sub-linear approach is preferred because errors are usually not permitted; unlike, for example, data mining, machine learning and optimization. In addition, storage is relatively cheap so the additional overhead incurred is not prohibitive in practice.

Structured encryption. Structured sub-linear search on encrypted data is formally captured by the notion of *structured encryption* (StE) which we define below. Informally, a StE scheme encrypts a data structure in such a way that it can be queried without revealing any useful information about the data or the query. As we will see throughout the course, StE can be constructed using a variety of cryptographic primitives. Moreover, we will also see that StE is a generalization of several other cryptographic primitives.

2 Data Structures

Since data structures are such an important component of search algorithms, we review some basic definitions.

Abstract data types. An *abstract data type* is a collection of objects together with a set of operations defined on those objects. Examples include sets, dictionaries (also known as key-value stores or associative arrays) and graphs. The operations associated with a data type fall into one of two categories: query operations, which return information about the objects; and update operations, which modify the objects. If the data type supports only query operations it is *static*, otherwise it is *dynamic*. For simplicity we define data types as having a single operation but note that the definitions can be extended to capture multiple operations in the natural way.

Data structures. A *data structure* for type a $\mathscr{T}$ is a concrete representation of type-$\mathscr{T}$ objects in some computational model. For us, the underlying model will always the random access machine (RAM) model of computation. The structure used to represent a type-$\mathscr{T}$ object is usually optimized to support the queries associated with $\mathscr{T}$ as efficiently as possible; that is, one designs the structure in such a way that there is an efficient algorithm to evaluate these queries. For data types that support multiple queries, the structure is usually designed to efficiently support as many of $\mathscr{T}$'s queries as possible. As a concrete example, the dictionary data type can be represented using various data structures depending on which queries one wants to support efficiently: hash tables support Get and Put in expected $O(1)$ time whereas balanced binary search trees support both operations in worst-case $\log(n)$ time. We model a type-$\mathscr{T}$ data structure as a collection of three ensembles $\mathbb{S} = \{\mathbb{S}_k\}_{k\in\mathbb{N}}$, $\mathbb{Q} = \{\mathbb{Q}_k\}_{k\in\mathbb{N}}$ and $\mathbb{A} = \{\mathbb{A}_k\}_{k\in\mathbb{N}}$. If a type-$\mathscr{T}$ structure DS supports only a single query, we often write $\mathsf{DS}(q)$ to denote the answer $a \in \mathbb{A}_k$ that results from querying DS on $q \in \mathbb{Q}_k$.

Basic data types. We will make use of several basic data types including arrays or random access memory (RAM), dictionaries and multi-maps. An array RAM of capacity n stores n items at locations 1 through n and supports read and write operations in $O(1)$ time. We write $v := \mathsf{RAM}[i]$ to denote reading the item at location i and $\mathsf{RAM}[i] := v$ the operation of storing an item at location i. A dictionary DX of capacity n is a collection of n label/value pairs $\{(\ell_i, v_i)\}_{i\leq n}$ and supports get and put operations. We write $v_i \leftarrow \mathsf{DX}[\ell_i]$ to denote getting the value associated with label ℓ_i and $\mathsf{DX}[\ell_i] := v_i$ to denote the operation of associating the value v_i in DX with label ℓ_i. A multi-map of capacity n is a collection of n label/tuple pairs $\{(\ell_i, V_i)_i\}_{i\leq n}$ that supports get and put operations. Like dictionaries, we write $V_i := \mathsf{MM}[\ell_i]$ to denote getting the tuple associated with label ℓ_i and $\mathsf{MM}[\ell_i] := V_i$ to denote the association of the tuple V_i with label ℓ_i. Multi-maps are the data type instantiated by inverted indices. In the encrypted search literature multi-maps are sometimes referred to as t-sets, tuple-sets and even as databases.

Notational abuse. As is usually done throughout computer science, we use notation that sometimes blurs the distinction between data types and data structures and trust that the reader can distinguish between the two based on context.

3 Syntax

As discussed above, a StE scheme encrypts a data structure in such a way that it can be privately queried. We now describe the syntax of a StE scheme.

Definition 3.1 (Structured encryption (response revealing)). *A single-round response-revealing structured encryption scheme* $\mathsf{STE} = (\mathsf{Setup}, \mathsf{Token}, \mathsf{Query})$ *for structures of type* $\mathscr{T}$ *consists of three polynomial-time algorithms that work as follows:*

- $(K, \mathsf{EDS}) \leftarrow \mathsf{Setup}(1^k, \mathsf{DS})$*: is a probabilistic algorithm that takes as input a security parameter* 1^k *and a structure* $\mathsf{DS} \in \mathbb{S}_k$ *and outputs a secret key* K *and an encrypted structure* EDS*.*

- $\mathsf{tk} \leftarrow \mathsf{Token}(K, q)$*: is a (possibly) probabilistic algorithm that takes as input a secret key* K *and a query* $q \in \mathbb{Q}_k$ *and returns a token* tk*.*

- $a \leftarrow \mathsf{Query}(\mathsf{EDS}, \mathsf{tk})$*: is a deterministic algorithm that takes as input an encrypted structure* EDS *and a token* tk *and outputs an answer* $a \in \mathbb{A}_k$*.*

We say that STE *is correct if for all* $k \in \mathbb{N}$*, for all* $\mathsf{DS} \in \mathbb{S}_k$*, for all* (K, EDS) *output by* $\mathsf{Setup}(1^k, \mathsf{DS})$ *and all sequences of* $m = \mathsf{poly}(k)$ *queries* $q_1, \ldots, q_m$*, for all tokens* tk_i *output by* $\mathsf{Token}(K, q_i)$*,* $\mathsf{Query}(K, \mathsf{tk}_i) = \mathsf{DS}(q)$*.*

StE schemes are typically used as follows. In the setup phase, the client computes $(K, \mathsf{EDS}) \leftarrow \mathsf{Setup}(1^k, \mathsf{DS})$, sends EDS to an untrusted server and keeps K private. During the query phase, the client computes and sends $\mathsf{tk} \leftarrow \mathsf{Token}(K, q)$ to the server who in turn computes $a \leftarrow \mathsf{Query}(\mathsf{EDS}, \mathsf{tk})$.

Response-hiding StE. We can define many variants of StE including interactive StE where the query phase requires multiple rounds of interaction; and *response-hiding* StE where the Query algorithm does not return answers in plaintext. We describe response-hiding schemes because they will be useful in future Lectures.

Definition 3.2 (Structured encryption (response-hiding))**.** *A single-round response-hiding structured encryption scheme* $\mathsf{STE} = (\mathsf{Setup}, \mathsf{Token}, \mathsf{Query}, \mathsf{Resolve})$ *for structures of type* $\mathscr{T}$ *consists of four polynomial-time algorithms such that* Setup *and* Token *are as in Definition 3.1 and that* Query *and* $\mathsf{Resolve}$ *works as follows:*

- $c \leftarrow \mathsf{Query}(\mathsf{EDS}, \mathsf{tk})$*: is a deterministic algorithm that takes as input an encrypted structure* EDS *and a token* tk *and outputs a message* c*.*
- $r \leftarrow \mathsf{Resolve}(K, c)$*: is a deterministic algorithm that takes as input a secret key* K *and a message* c *and outputs an answer* $a \in \mathbb{A}_k$*.*

We say that STE *is correct if for all* $k \in \mathbb{N}$*, for all* $\mathsf{DS} \in \mathbb{S}_k$*, for all* (K, EDS) *output by* $\mathsf{Setup}(1^k, \mathsf{DS})$ *and all sequences of* $m = \mathsf{poly}(k)$ *queries* $q_1, \ldots, q_m$*, for all tokens* tk_i *output by* $\mathsf{Token}(K, q_i)$*, for all messages* c_i *output by* $\mathsf{Query}(\mathsf{EDS}, \mathsf{tk}_i)$*,* $\mathsf{Resolve}(K, c_i) = \mathsf{DS}(q)$*.*

4 Security

Intuitively, we would like an StE scheme to guarantee that

> *the encrypted structure reveals no useful information about the underlying structure and that the tokens reveal no useful information about the underlying query.*

Four our purposes, we will consider "non-useful" any information that can be derived from the security parameter k. For example, this would include information about the structure or query spaces. Unfortunately, such a security notion seems hard to achieve efficiently so we would like to weaken it to allow for trade-offs. How much exactly we should weaken the definition depends on which trade-offs we are willing to make. At this point, we will not consider the question of which trade-offs are reasonable and which are not (that will come later). What we want is a definition that allows us to study notions of security that are "less than perfect". There are at least two approaches one could take to study weaker notions of security.

Usage-based. The first is to use a standard notion of security but make some assumption about how the cryptosystem will be used. For example, we saw in Lecture 2 that deterministic encryption (DtE) schemes cannot be CPA-secure. But what if we still want to study their security? One approach is to keep the notion of CPA-security as our target definition but study what happens when a DtE scheme is used on messages that come from high-min-entropy distributions. [1] Under this assumption, we can show that a DtE scheme is CPA-secure so

[1] Messages from high min-entropy distributions are, roughly, messages that are hard to guess.

this tells us something about its security. The limitation of this approach, however, is that it does not tell us anything about the security of DtE when messages not from high-min-entropy distributions and, unfortunately, this is the typical case.

Leakage-based. Another approach is to explicitly weaken the definition and try to understand the consequences. But how do we do this exactly? Let's consider the example of encryption. As we saw in the previous lecture, the notion of CPA-security provides the following guarantee,

> *the ciphertext reveals no useful partial information about the plaintext, even if the adversary has oracle access to an encryption oracle.*

We also saw how to formalize this using both game-based and simulation-based definitions. The latter formulation, in particular, required that whatever can be computed in PPT by an adversary given the ciphertext can also be computed in PPT by a simulator without the ciphertext.

To weaken this definition, we provide the simulator with some information about the message. We refer to this information as *leakage* and capture it with a stateful function $\mathcal{L}(m)$ of the message. Returning to our DtE example, we can define the leakage $\mathcal{L}$ on a tuple of messages $\mathbf{m} = (m_1, \ldots, m_t)$ underlying a tuple of DtE ciphertexts $\mathbf{ct} = (\mathrm{ct}_1, \ldots, \mathrm{ct}_t)$ as a $t \times t$ matrix M such that $M[i,j] = 1$ if $m_i = m_j$ and $M[i,j] = 0$ if $m_i \neq m_j$. As a result, the definition now requires that whatever can be computed in PPT by an adversary given the ciphertext can also be computed in PPT by a simulator that is given $\mathcal{L}(\mathbf{m}) = M$.

4.1 Formalizing Leaky Primitives

We now give a formal security definition for StE using the leakage-based approach. The definition is simulation-based and parameterized with *leakage functions* as described above. It is possible to define an analogous game-based definition but, unlike standard encryption, we do not know if the two notions are equivalent.

We give two definitions of security for StE. Both Definitions have a similar structure and are based on two experiments: a **Real** one and an **Ideal** one. In the **Real** experiment, an adversary interacts with the StE scheme. It generates a structure and various queries and receives an encrypted structure and tokens. In the **Ideal** experiment, on the other hand, the adversary interacts with a simulator. The adversary still generates a structure and queries but it receives a *simulated* encrypted structure and simulated tokens all generated by the simulator who itself is only given the leakage of the structure and queries. The main difference between the definitions is in how the adversary is allowed to choose its queries.

Non-adaptive security. The first definition is non-adaptive in the sense that the adversary restricted in how it generates its queries. Intuitively, this definition guarantees that $\mathcal{A}$ will not learn anything about the structure and queries beyond the leakages explicitly captured by $\mathcal{L}_{\mathsf{S}}$ and $\mathcal{L}_{\mathsf{Q}}$ which are stateful leakage functions. We refer to the pair $(\mathcal{L}_{\mathsf{S}}, \mathcal{L}_{\mathsf{Q}})$ as a *leakage profile*.

Definition 4.1 (Non-adaptive security). *Let* $\mathsf{STE} = (\mathsf{Setup}, \mathsf{Token}, \mathsf{Query})$ *be a structured encryption scheme for structures of type* $\mathscr{T}$ *and consider the following probabilistic experiments where* $\mathcal{A}$ *is a stateful adversary,* $\mathcal{S}$ *is a stateful simulator,* $\mathcal{L}_{\mathsf{S}}$ *and* $\mathcal{L}_{\mathsf{Q}}$ *are stateful leakage functions and* $z \in \{0,1\}^*$*:*

$\mathbf{Real}_{\mathsf{STE},\mathcal{A}}(k)$*: given* z *the adversary* $\mathcal{A}$ *outputs a structure* DS *and polynomially-many queries* $(q_1, \ldots, q_m)$*. It then receives* EDS *and* $(\mathsf{tk}_1, \ldots, \mathsf{tk}_m)$*, where* $(K, \mathsf{EDS}) \leftarrow \mathsf{Setup}(1^k, \mathsf{DS})$ *and for all* $1 \leq i \leq m$*,* $\mathsf{tk}_i \leftarrow \mathsf{Token}(K, q_i)$*. Finally,* $\mathcal{A}$ *outputs a bit* b *that is returned by the experiment.*

$\mathbf{Ideal}_{\mathsf{STE},\mathcal{A},\mathcal{S}}(k)$*: given* z *the adversary* $\mathcal{A}$ *outputs a structure* DS *and polynomially-many queries* $(q_1, \ldots, q_m)$*. Given* z*,* $\mathcal{L}_{\mathsf{S}}(\mathsf{DS})$ *and* $(\lambda_1, \ldots, \lambda_m)$*, where* $\lambda_i \leftarrow \mathcal{L}_{\mathsf{Q}}(\mathsf{DS}, q_i)$*, the simulator* $\mathcal{S}$ *gives* $\mathcal{A}$ *an encrypted data structure* EDS *and a tuple of tokens* $(\mathsf{tk}_1, \ldots, \mathsf{tk}_m)$*. Finally,* $\mathcal{A}$ *outputs a bit* b *that is returned by the experiment.*

We say that STE *is non-adaptively* $(\mathcal{L}_{\mathsf{S}}, \mathcal{L}_{\mathsf{Q}})$*-secure if there exists a* PPT *simulator* $\mathcal{S}$ *such that for all* PPT *adversaries* $\mathcal{A}$ *and all* $z \in \{0,1\}^*$*,*

$$\left|\Pr\left[\,\mathbf{Real}_{\mathsf{STE},\mathcal{A}}(k) = 1\,\right] - \Pr\left[\,\mathbf{Ideal}_{\mathsf{STE},\mathcal{A},\mathcal{S}}(k) = 1\,\right]\right| \leq \mathsf{negl}(k).$$

Note that in the experiments the adversary outputs its structure DS and queries $(q_1, \ldots, q_m)$ before it sees the encrypted structure EDS and tokens $(\mathsf{tk}_1, \ldots, \mathsf{tk}_m)$. In particular, this means that it cannot choose its queries as a function of the encrypted structure and the tokens. What this means exactly is not completely clear and is reminiscent of the situation in secure multi-party computation described at a high-level in the Damgard paper.

Adaptive security. In the second definition, we do not restrict how the adversary generates its queries. It is allowed to choose them as a function, not only of previous queries and responses, but also of the encrypted structure and tokens.

Definition 4.2 (Adaptive security). *Let* $\mathsf{STE} = (\mathsf{Setup}, \mathsf{Token}, \mathsf{Query})$ *be a structured encryption scheme for type* $\mathscr{T}$ *and consider the following probabilistic experiments where* $\mathcal{A}$ *is a stateful adversary,* $\mathcal{S}$ *is a stateful simulator,* $\mathcal{L}_{\mathsf{S}}$ *and* $\mathcal{L}_{\mathsf{Q}}$ *are leakage profiles and* $z \in \{0,1\}^*$*:*

$\mathbf{Real}_{\mathsf{STE},\mathcal{A}}(k)$*: given* z *the adversary* $\mathcal{A}$ *outputs a structure* DS *of type* $\mathscr{T}$ *and receives* EDS *from the challenger, where* $(K, \mathsf{EDS}) \leftarrow \mathsf{Setup}(1^k, \mathsf{DS})$*. The adversary then* adaptively *chooses a polynomial number of queries* $q_1, \ldots, q_m$*. For all* $i \in [m]$*, the adversary receives* $\mathsf{tk}_i \leftarrow \mathsf{Token}(K, q_i)$*. Finally,* $\mathcal{A}$ *outputs a bit* b *that is output by the experiment.*

$\mathbf{Ideal}_{\mathsf{STE},\mathcal{A},\mathcal{S}}(k)$*: given* z *the adversary* $\mathcal{A}$ *generates a structure* DS *of type* $\mathscr{T}$ *which it sends to the challenger. Given* z *and leakage* $\mathcal{L}_{\mathsf{S}}(\mathsf{DS})$ *from the challenger, the simulator* $\mathcal{S}$ *returns an encrypted data structure* EDS *to* $\mathcal{A}$*. The adversary then* adaptively *chooses a polynomial number of operations* $q_1, \ldots, q_m$*. For all* $i \in [m]$*, the simulator receives query leakage* $\mathcal{L}_{\mathsf{Q}}(\mathsf{DS}, q_i)$ *and returns a token* tk_i *to* $\mathcal{A}$*. Finally,* $\mathcal{A}$ *outputs a bit* b *that is output by the experiment.*

We say that STE *is adaptively* $(\mathcal{L}_{\mathsf{S}}, \mathcal{L}_{\mathsf{Q}})$*-secure if there exists a* PPT *simulator such that for all* PPT *adversaries* $\mathcal{A}$ *and all* $z \in \{0,1\}^*$*,*

$$\left|\Pr\left[\,\mathbf{Real}_{\mathsf{STE},\mathcal{A}}(k) = 1\,\right] - \Pr\left[\,\mathbf{Ideal}_{\mathsf{STE},\mathcal{A},\mathcal{S}}(k) = 1\,\right]\right| \leq \mathsf{negl}(k).$$

Limitations. Note that these definitions *do not tell us anything about what can happen as a consequence of this leakage*. In particular, this means that a scheme that is adaptively or non-adaptively $(\mathcal{L}_{\mathsf{S}}, \mathcal{L}_{\mathsf{Q}})$-secure for some leakage profile could be completely broken if one can exploit the leakage effectively. What the definition does tell us, however, is that the construction does not have any design flaws beyond the leakage (and any underlying computational assumptions). In spirit, this is in line with the traditional provable/reductionist security paradigm where we prove that a scheme is secure under some computational assumption. The latter does not tell us whether a scheme is truly secure or not but it does "reduce the attack surface" of the scheme by informing us that cryptanalytic effort can focus on the underlying assumption and not the higher-level design. Similarly, these definitions tell us that cryptanalytic effort should be focused on exploiting the leakage.

References

[1] Ronitt Rubinfeld. Sublinear time algorithms. See http://people.csail.mit.edu/ronitt/sublinear.html.