2014

DNSSEC -- authenticated denial of existence: understanding zone enumeration

Vasant, Sachin

https://hdl.handle.net/2144/15138

Boston University
DNSSEC – AUTHENTICATED DENIAL OF EXISTENCE:
UNDERSTANDING ZONE ENUMERATION

by

SACHIN VASANT

M.Sc. Anna University (India), 2012

Submitted in partial fulfillment of the
requirements for the degree of
Master of Science
2014
Approved by

First Reader

Leonid Reyzin, PhD
Associate Professor of Computer Science

Second Reader

Sharon Goldberg, PhD
Assistant Professor of Computer Science
Acknowledgments

I would like to thank Sharon Goldberg, Dimitrios Papadopolous and Leonid Reyzin. This entire thesis is based on joint work with them, and any progress in this regard would have been impossible without them. I would like to thank other members of BUSec for the numerous discussions, both on this topic and otherwise. These discussions were instrumental in my learning experience, in all aspects of cryptography and network security.
DNSSEC – AUTHENTICATED DENIAL OF EXISTENCE :
UNDERSTANDING ZONE ENUMERATION
SACHIN VASANT
ABSTRACT

Over the years DNS has proved to be an integral part of the internet infrastructure. For our purposes, DNS is simply a large scale distributed database that maps human-readable domain names to network recognizable IP addresses. Unfortunately, authenticity of responses was not integral to the initial DNS design. This lead to the possibility of a very practical forgery of responses as displayed by Kaminsky’s cache poisoning attacks. DNSSEC is primarily designed as a security extension of DNS, that guarantees authenticity of DNS responses.

To answer invalid queries in an authenticated manner, DNSSEC initially employed the NSEC records. To its credit, NSEC allowed nameservers to precompute signatures for such negative responses offline. As a result, NSEC is highly scalable while preserving the authenticity/correctness of responses. But, while doing so, NSEC leaks domains from nameserver’s zone. This is called zone enumeration.

To counter zone enumeration, NSEC3 was deployed. It is a hashed authenticated denial of existence of mechanism, i.e., it reveals the hashes of the zones in a domain. NSEC3 yet allows offline signatures, and is scalable like NSEC. Unfortunately, hashes are vulnerable to dictionary attacks a property exploited by conventional NSEC3 zone enumeration tool, e.g., nsec3walkertool.

This leads us to investigate the possibility of constructing an authenticated denial of existence of mechanism which yet allows offline cryptography. To do so, we first define the security goals of a “secure” DNSSEC mechanism in terms of an Authenticated Database System (ADS) with additional goals of privacy, that we define. Any protocol that achieves these goals, maintains the integrity of DNSSEC responses and prevents zone enumeration. We then show that any protocol that achieves such security goals, can be used to con-
struct weak signatures that prevent selective forgeries. This construction, though a strong indication, doesn’t confirm the impossibility of generating proofs offline.

To confirm that such proofs aren’t possible offline, we show attacks of zone enumeration on two large classes of proofs. The provers/responders in this case either repeat proofs non-negligibly often or select proofs as subsets from a pre-computed set of proof elements. The attackers we present use a dictionary of all elements that are likely to occur in the database/zone. The attackers prune the said dictionary to obtain the set of all elements in the database (along with a few additional elements that are erroneously classified to be in the database). These attackers minimize the number of queries made to such responders and are loosely based on the paradigm of Probably Approximately Correct learning as introduced by Valiant.
# Contents

1 Introduction .......................... 1

1.1 DNSSEC ................................ 2

1.2 Authenticated Denial of Existence – Zone Enumeration ............... 5

1.3 Outline ................................ 7

2 Defining the Security of DNSSEC .................................. 9

2.1 Candidate Definitions ........................................ 10

2.1.1 Motivation ........................................ 10

2.1.2 Modeling Integrity .................................. 11

2.1.3 Modeling Privacy .................................. 13

2.2 The DNSSEC standard and NSEC3 ................................... 15

2.3 Pruned Merkle Tree – Hybrid Scheme ............................... 17

2.4 Signatures from DNSSEC ..................................... 19

2.4.1 Signature Definitions .................................. 20

2.4.2 Construction ........................................... 20

2.4.3 Security Proof ....................................... 21

2.4.4 Offline? .............................................. 23

3 Defining Zone Enumeration ...................................... 25

3.1 Defining Privacy ............................................ 25

3.2 Oblivious Attacks .............................................. 28

3.2.1 NSEC3 ................................................ 28

3.2.2 Hybrid Scheme ....................................... 32

vi
4  Oblivious Attacks and Epsilon Nets  

4.1  Preliminaries  

4.2  Polynomial Proofs  

4.2.1  Relation to Epsilon Nets  

4.3  Subset Proofs  

4.3.1  Learner and epsilon-nets  

4.3.2  Pruner and II  

Appendix  

Bibliography
List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>A sample resolution of <a href="http://www.bu.edu">www.bu.edu</a></td>
<td>2</td>
</tr>
<tr>
<td>1.2</td>
<td>A sample resolution of <a href="http://www.bu.edu">www.bu.edu</a> based on [8]</td>
<td>3</td>
</tr>
<tr>
<td>4.1</td>
<td>A graphical representation of repeated proofs</td>
<td>41</td>
</tr>
<tr>
<td>4.2</td>
<td>A graphical representation of subset proofs</td>
<td>45</td>
</tr>
</tbody>
</table>
List of Abbreviations

ADS . . . . Authenticated Database Systems
DNS . . . . Domain Name System
DNSSEC Domain Name System Security Extensions
KSK . . . Key-signing key
RR . . . Resource Record
RRset . . A set of RRs of a given type
TLD . . . Top-level Domains
ZSK . . . Zone-signing key
Chapter 1

Introduction

The domain name system (DNS) \cite{25, 26} is an integral part of the modern Internet architecture. Users intend to communicate with remote servers, which they identify using descriptive “domain names” like \texttt{foo.org}. They use the DNS to resolve the IP addresses of the server hosting the said domain name.

The DNS system primarily involves three parties,

Client  The client is the end user in the system that wishes to have a domain name resolved. Thus, it initiates a DNS lookup query.

Recursive Resolvers  The recursive resolvers/recursors interface with the clients. They are typically in the client’s network, but there are currently open resolvers too, spread across the Internet. The resolvers receive queries from the clients, verify their cache for responses. If not found, they resolve the query on behalf of the clients.

Nameservers  The nameservers host the DNS data, \textit{i.e.}, the actual mapping between the domain names and the IP addresses (among other data related to the said domain names). The resolvers query the nameservers until they obtain the right name server to return the required information corresponding to a domain.

To resolve names, the DNS system maintains a hierarchy of nameservers. Each nameserver contains a \textit{zone}, which is its set of domains and referrals. The root nameservers are at the top of the hierarchy and store the root zone, followed by the top-level domains (TLDs) like \texttt{.org, .edu} etc. and the country-level TLDs (ccTLDs) like \texttt{.in, .cn} etc. These are followed by more fine-grained zones, for example \texttt{foo.org, bu.edu}. 
Data in the zone is stored as resource records (RRs). A nameserver is authoritative for a zone, if the details in the zone are consistent with what is generated at the actual source of the zone. Some of the typical RRs are the A record which stores the IPv4 address corresponding to a specified url, the AAAA record which stores the IPv6 address. The NS record holds the nameserver to be referred to for the given the query. Some zones are replicated in multiple servers. One of the servers is known as the master server for the zone, and the others are the slaves. All updates are done at the master, and the zones of the slaves are updated using zone-transfer protocols (AXFR).

The said information is used to lookup the domain required. The resolution of \texttt{www.bu.edu} is illustrated in the figure. The query texts in the figure are based on the texts in [8].

![Diagram showing a sample resolution of www.bu.edu](image)

Figure 1.1: A sample resolution of \texttt{www.bu.edu}

1.1 DNSSEC

Note that the DNS doesn’t have authentication integrated in its design. Thus, the resolvers and clients have been susceptible to cache poisoning attacks like the Kaminsky bug [18]. Such attacks accentuated the need for authenticated DNS responses.
The DNS Security Extensions (DNSSEC) first introduced in 1999 [3], extends the DNS to provide authenticated DNS responses. DNSSEC [5, 7, 6] aims to provide authentic responses from the nameservers to the resolvers. The goal of authenticated delegation of zones from masters to slaves, and authentic responses from the resolver to the client are not in the purview of DNSSEC.

DNSSEC adds additional resource records, primarily, the DNSKEY, RRSIG and the DS records. The DNSKEY records primarily holds the keys used to sign the zones (zone signing keys, ZSK) or to sign the set of all DNSKEYs held in the zone (the key signing keys, KSK). DS records are held by nodes higher up in the hierarchy. The DS record is the hash of a KSK of the child node. RRSIG records consist of signatures on a particular type of resource record set (RRset). An extension of the above lookup process using DNSSEC is mentioned in figure 1.2: The verification of the resolution process entails the following steps:

1. The recursive resolver verifies the signature corresponding to the received RRSIG for
using the corresponding DNSKEY record. If valid, it then verifies the RRSIG corresponding to the said DNSKEY RRset.

2. The recursor then verifies if the DNSKEY is indeed correct using the DS record retrieved from edu. The signature corresponding to this DS is verified using the DNSKEYs of edu. The signature corresponding to the DNSKEY RRset is also verified.

3. To verify if the DNSKEYs were indeed that of edu, the recursor verifies the DS record from the root. The signature corresponding to this DS record is verified using the root’s DNSKEY. The signature corresponding to the DNSKEY RRset is also verified.

Note that in the above recursion, the resolver doesn’t validate the DNSKEYs of the root. In the true sense, it does validate the DNSKEYs. Here, the DNSKEYs of the root are stored as trusted anchors in its initially configured cache. The root in DNSSEC terms becomes a trusted anchor, above which validation isn’t necessary. The chain of servers up to that the server validates all the way up to the trusted anchor is the authentication chain. In case of partial deployment not all nameservers will be able to form an authentication chain up to a trust anchor, thus forming an island of security. The recursive resolver can set these name servers lower in the hierarchy as trusted anchors too, as long as it is certain of the credentials of the keys it receives for these nameservers. For further details on trusted anchors, we refer the readers to [5].

Note that every DNSSEC response requires at least one additional RRSIG record. Most responses may also receive an accompanying DNSKEY. Therefore, the size of the responses already exceed the accepted DNS packet lengths. Thus, DNSSEC requires the used of the extended DNS, EDNS (defined in [34]). Thus, the length of DNSSEC responses is an important design constraint, and sizes exceeding 4096 bits is considered disadvantageous.
1.2 Authenticated Denial of Existence – Zone Enumeration

We have earlier described the lookup procedure for a valid query that must exist in a zone. Indeed, invalid queries need to be handled in the same vein too. In RFCs [5, 7, 8], a new record NSEC (next secure) was proposed for the purpose. Suppose we ask an invalid query \texttt{y.foo.org} to the nameserver hosting \texttt{foo.org}. The name server retrieves the two valid domains in its zone that are closest to the query (say \texttt{x.foo.org} and \texttt{z.foo.org}) and return the same as a NSEC record, accompanied with an RRSIG. Such NSEC records and their RRSIGs can be pre-computed, since there are only finitely many such records per zone.

A disadvantage with the NSEC records were, that a few adversarially generated invalid queries could reveal the entire zone. Such an attack is called \textit{zone enumeration}. In DNS, the issue doesn’t arise, because, we have to explicitly query for an entity in the zone, to obtain information about it. Even though DNSSEC doesn’t aim to provide confidentiality [5], the zone enumeration is yet an issue. To quote [22],

An enumerated zone can be used, for example, as a source of probable e-mail addresses for spam, or as a key for multiple WHOIS queries to reveal registrant data that many registries may have legal obligations to protect. Many registries therefore prohibit the copying of their zone data; however, the use of NSEC RRs renders these policies unenforceable.

The issue could be mitigated by on-line signing. But on-line signing comes with a few caveats. A few security concerns with on-line signing (quoting [35] verbatim) are:

1. On-demand signing requires that a zone’s authoritative servers have access to its private keys. Storing private keys on well-known Internet-accessible servers may make them more vulnerable to unintended disclosure.

2. Since generation of digital signatures tends to be computationally demanding, the requirement for on-demand signing makes authoritative servers vulnerable to a denial
of service attack.

3. If the responses are predictable, on-demand signing may enable a chosen-plaintext attack on a zone’s private keys. Zones using this approach should attempt to use cryptographic algorithms that are resistant to chosen-plaintext attacks. It is worth noting that although DNSSEC has a "mandatory to implement" algorithm, that is a requirement on resolvers and validators – there is no requirement that a zone be signed with any given algorithm.

Taking the above caveats into account, the rfc [22] proposes NSEC3 for DNSSEC. The NSEC3 resource record returns a cryptographic hashes instead of the actual domains for a queried invalid domain. So again, if we ask an invalid query \texttt{y.foo.org} to the nameserver hosting \texttt{foo.org}, it returns $h_1$ and $h_2$ (corresponding to the hash of two values say \texttt{m.foo.org} and \texttt{b.foo.org} in the zone), such that $h_1 < \text{hash(y.foo.org)} < h_2$. This NSEC3 is of course accompanied by its RRSIG record. We have further details on the same in the later chapters.

Note that, even in this case, zone-enumeration is yet possible although with a lot more effort as compared to NSEC, as long as we obtain a suitable dictionary. In fact, it is yet easier to enumerate a zone than in the case of DNS, because DNS requires us to query on every individual element that we suspect lies in the zone. This has been exploited in the nsec3walker tool [2]. In our work, we again illustrate a theoretically feasible attack on the NSEC3. The attack in our work is for illustrative purposes only, to show why NSEC3 will not achieve the security goals for privacy that we define.

The above caveats notwithstanding, there have been calls for online signing nevertheless. There have been DNSSEC tools like the PowerDNSSEC [17] and Phreebird [19] that perform signing online. Also, using all-but-one signatures, one can do online signing for invalid queries [15]. In this case, even in case of key-compromise, the adversary cannot sign non-existence records for entities that are actually in the zone.

But, an offline authenticated denial of existence mechanism is yet desirable, especially
to minimize the computational load on the servers. Thus, in our work, we examine the possibility of the existence of such a desirable authenticated denial of existence mechanism. For some large classes of functions, we are able launch zone enumeration attacks, thus negating the possibility of denial mechanisms from such classes of functions.

1.3 Outline

Given that NSEC3 doesn’t prevent zone enumeration, it is interesting to verify if there exists any protocol that can prevent zone enumeration while delegating all computations using trapdoors offline. We think the answer for the same is in the negative. Our investigations take two broad approaches:

1. We initially investigate the possibility of a reduction to cryptographic signatures. The idea for the same would thus be to claim that if there exists a “secure” DNSSEC that does not use any trapdoors in its online computation, then there exists signatures which does only public key computations online. To do so, we first formally define the security goals of DNSSEC using authenticated databases ([31]) and add our definition of privacy to it (semantic security). Then we show that the same could be used to construct signatures with weak unforgeability guarantees. The limitations in the unforgeability guarantees stem from the fact that privacy in this case is limited to protecting the database/zone and not the status of the entire space of domains.

2. We define two large classes of proof-systems that we think are the most logical methods to avoid online cryptography. First, there exists only a small set of proofs, each of which is repeated non-negligibly often. This is a direct generalization of NSEC3. The second class, uses a bounded set of proof components, where every proof is subset of this set of proof components. We show that there exists “efficient” zone enumeration adversaries (that use dictionaries) for both these classes of proof systems.

The following chapters essentially elaborate on the aforementioned approaches. Specifically,
Chapter 2. We define the security goals of DNSSEC using Authenticated databases in section 2.1. We describe the current DNSSEC standard (with NSEC3) in the framework of the said definition is section 2.2. To add to the same, we construct what an offline version of a standard construction for zero-knowledge databases [10] in section 2.3. In section 2.4 we construct such signatures with weak unforgeability guarantees from such “secure” DNSSEC protocols.

Chapter 3. Chapter 3 initiates our attempt with approach 2. We look at certain types of zone enumeration adversaries. In section 3.1, we present two classes of attackers, the oblivious attacker and the adaptive attacker. These attackers are loosely based on uniform PAC learners. We then present oblivious attacks against NSEC3 and against the aforementioned construction from zero-knowledge databases. The attacks against NSEC3 are already well-known. We just fit the attack into our formal notion of zone-enumeration adversaries to analyse its correctness and efficiency.

Chapter 4. In chapter 4, our goal is to illustrate oblivious attacks on the two large classes of proof-systems. To do so, we require sampling tools, for example the epsilon nets. In section 4.1, we present the requisite preliminaries. In section 4.2, we formally state the class of functions that has bounded number of proofs, and hint on the attack using the oblivious adversary. In section 4.3, we formally state the class of functions that generate proofs from a set of proof elements, and state our attack on the same.
Chapter 2

Defining the Security of DNSSEC

We look at every stage of interaction in DNSSEC as to having three parties. The authenticator is authoritative to the zone and holds the signing key for the zone, most likely held in the master server of the zone. In other words, the authenticator is responsible for generating proofs for the zone, and is the “offline” component of the DNSSEC component. The responder is the online component of the authoritative nameservers (including the slave servers). They generate the required DNSSEC responses for every queries. The recursive resolvers or the end clients (if they use the CD bit) that validate the DNSSEC responses are the verifiers in the system\(^1\). Note that in the chapter we don’t consider the whole recursive query process. We assume that the DNSKEYs are a robust key distribution mechanism and abstract the exchange of DNSKEY. We consider only the interaction between the validator and the nameserver that is authoritative for the zone.

In this chapter, we attempt to formally define the security goals of DNSSEC. We represent the conventional DNSSEC-NSEC3 and another plausible “candidate” replacement for the same using the definition, as examples. We don’t prove it’s security, because, we show both protocols aren’t private in chapter 3. Further, we show that if a protocol achieves the security goals of DNSSEC then we can construct a weak signature using the same.

\(^1\)We borrow the notations authenticator, responder and verifier from [31].
2.1 Candidate Definitions

2.1.1 Motivation

In the DNSSEC scenario, it is in the authenticator's interest to ensure the client gets correct responses, so we assume it is trusted. Thus, in our definitions of security, we will assume the authenticator is always honest, and that the verifier receives the correct public key from the owner. The responder, on the other hand, can be malicious, or may have an attacker modify the information it gets from the authenticator. There may also be a man-in-the-middle (cache poisoning) adversary that forges malicious responses. Thus, the protocol needs to be designed so as to allow the verifier to detect incorrect answers given by a malicious host or modified in transit by the adversary. These requirements are identical to the problem of authenticated data structures/outsourced databases (for example, [31]).

However, we have one more requirement: we are concerned that the verifier may also be malicious and may try to obtain more information than intended by the protocol. Specifically, we want to prevent the verifier from obtaining entries that it has not requested (for example “zone enumeration” with NSEC). This secrecy requirement makes the problem similar to the problem of Zero-Knowledge Sets (or Zero-Knowledge Databases) [24], which protect against malicious verifiers. However, Zero-Knowledge Sets do not distinguish between the authenticator and the responder (there are only two parties: prover–authenticator+responder and verifier), so we cannot use the Zero-Knowledge definition unmodified. In fact, our task is easier, because having a trusted authenticator (and, therefore, an honestly generated public key) makes it easier to guarantee consistency of responses.

We will use the definition of authenticated data structures from [31] as our starting point, and will consider different secrecy requirements in addition to it.
2.1.2 Modeling Integrity

We model the correctness requirements of DNSSEC as an authenticated database. We use a few notations from [31], but present a more generalised definition to fit our purposes.

For our purposes, a database \( D : \{0,1\}^* \rightarrow \{0,1\}^* \cup \{Out\} \) is a mapping that, for an arbitrary input string, gives either an output string, or a special indicator \( Out \) that means that the input is not in the database. An authenticated database scheme allows the owner of a database \( D \) to authenticate \( D \) and outsource the same to another server. This server then responds to queries on \( D \) by other clients. Since, we require integrity guarantees of responses, we define completeness and soundness as security properties any authenticated database scheme must satisfy. First, we formally define an authenticated database scheme.

**Definition 1.** An authenticated database scheme (ADS) is a quadruple of PPT algorithms \((\text{Gen}, \text{Auth}, \text{Res}, \text{Ver})\) such that:

**Key Generation** Algorithm \( \text{Gen} \) run by the owner takes as input a security parameter \( 1^k \), and outputs a key pair. We write \((PK, SK) \leftarrow \text{Gen}(1^k)\).

**Authenticator** \( \text{Auth} \) upon receiving the public key \( PK \), the secret key \( SK \) and the database \( D \) as input, halts after outputting a commitment/authentication string \( Comm \) and a committed database \( CD \). We denote the same as \((Comm, CD) \leftarrow \text{Auth}(PK, SK, D)\).

**Responder** The responder \( \text{Res} \) on receiving the public key \( PK \) and the committed database \( CD \) from the authenticator, as well as a query \( x \) from the client, produces a proof \( \pi_x \) on \( D(x) \), to be returned to the client. We denote the same by \( \pi_x \leftarrow \text{Res}(CD, PK, x)\).

**Verifier** The verifier algorithm \( \text{Ver} \) receives the proof \( \pi_x \) on \( x \), verifies it using \( PK \) and \( Comm \), and outputs \( y = D(x) \) if it is valid, or \( y = Invalid \) otherwise. We denote the same as \( y \leftarrow \text{Ver}(x, \pi_x, PK, Comm) \). (Note that we view \( \pi_x \) as carrying information \( D(x) \) in it.)
Our definition is deliberately as general as possible, so as to allow for many options in designing DNSSEC. In particular, the system may be easier to use if \( PK \) is simply a public key, independent of \( D \) and reusable for multiple databases. This possibility is within the scope of the definition, because \( \text{Gen} \) depends only on the security parameter.

\( \text{Comm} \) is a “digest” on the database (e.g., signed root of a Merkle hash tree [23]), but the requirement of such a digest is not enforced. If the ADS doesn’t require a digest, \( \text{Comm} \) can be assigned a special symbol \( \perp \). The database digest may be sent to the verifier directly, or through the responder (in which case, according to our definition, it will be part of every proof \( \pi_x \)). Note that \( \text{Res} \) can be stateful, so proofs can have a reusable precomputed component based on \( PK \) and \( \text{CD} \), reused for multiple queries \( x \), if that helps efficiency.

To model correctness, an authenticated database scheme must satisfy the property of completeness. Informally, completeness simply states that if \( \text{Gen, Auth and Res} \) function as in the definition, then verification of the proof produced by \( \text{Res} \) would give the expected output with overwhelming probability. More formally,

**Definition 2.** An authenticated database \((\text{Gen, Auth, Res, Ver})\) is \( \epsilon \)-complete if \((PK, SK) \leftarrow \text{Gen}(1^k), (\text{Comm, CD}) \leftarrow \text{Auth}(PK, SK, D) \) and \( \pi_x \leftarrow \text{Res}(CD, PK) \) for any database \( D \), then

\[
\Pr[D(x) \leftarrow \text{Ver}(x, \pi_x, PK, \text{Comm})] \geq 1 - \epsilon
\]

Now, that we have defined the correctness of the scheme, we need to model integrity. Note that, for integrity we require that a verifier can detect fake responses generated by a malicious responder or some man-in-the-middle. We capture this property formally through a weak definition of soundness.

**Definition 3.** For a given database \( D \), the authenticated database \((\text{Gen, Auth, Res, Ver})\) is \((\tau, \epsilon)\)-sound, if every adversary \( \text{Res} \) with running time at most \( \tau \), has probability at most \( \epsilon \) in winning the following game:

- \((PK, SK) \leftarrow \text{Gen}(1^k)\)
• \((Comm, CD) \leftarrow \text{Auth}(PK, SK, D)\)

• \((x, \pi_x) \leftarrow \text{Res}(Comm, CD, PK)\)

where \text{Res} wins if \(y \leftarrow \text{Ver}(x, \pi_x, Comm, PK)\) such that \(y \neq \text{Invalid}\) and \(y \neq D(x)\), i.e., the proof \(\pi_x\) on \(x\) is not consistent with the database mapping, but is yet valid.

The ADS is then considered sound if for every \(\tau \approx \text{poly}(k)\), there exists an \(\epsilon \approx \text{negl}(k)\), such that ADS is \((\tau, \epsilon)\)-sound.

Note the definition of soundness assumes that the verifier receives the correct \(PK\) and correct commitment \(Comm\); no guarantees are provided if that is not the case. There can be a variety of mechanisms (most likely, using certification and digital signatures) for ensuring this property, but they are independent of the security of the authenticated database scheme itself. If there arises a scenario where the “digest” generated using \text{Auth} is subject to modification by some \text{Res}, the “digest” can be added as a re-usable component of \(\pi_x\) and \(Comm\) be set to \(\bot\).

Man-in-the-middle adversaries are weaker adversaries than the ones modeled in definition 3. For such adversaries, we can follow the same game as above, with the difference that, a man-in-the-middle doesn’t receive CD as input.

2.1.3 Modeling Privacy

Now, that we have defined an ADS, we would like to define potential privacy goals for ADS. Some of the candidate definitions follow.

Firstly, the most natural definition of privacy is that of zero-knowledge. For the same, we make use of the definition zero-knowledge from [10].

**Definition 4. Zero-knowledge.** There exists a simulator \(\text{Sim} = (\text{Sim}_1, \text{Sim}_2, \text{Sim}_3)\) such that for probabilistic polynomial-time malicious verifiers \(\overline{\text{Ver}} = (\overline{\text{Ver}}_1, \overline{\text{Ver}}_2)\), the absolute value of the difference

\[
\Pr[(PK, SK) \leftarrow \text{Gen}(1^k) ; (D, \text{state}_V) \leftarrow \overline{\text{Ver}}_1(1^k, PK) ; \]

\[
\]
\((Comm, CD) \leftarrow \text{Auth}(PK, SK, D) \text{ s.t. } \overline{\text{Ver}}_{2}^{\text{Res}(CD, PK, \cdot)}(state_V) = 1\) – 
\[
\text{Pr}[(PK, state_o) \leftarrow \text{Sim}_1(1^k) ; (D, state_V) \leftarrow \overline{\text{Ver}}_1(1^k, PK) ; (Comm, state_S) \leftarrow \text{Sim}_2(1^k, PK, state_o) \text{ s.t. } \overline{\text{Ver}}_2^{\text{Sim}_3(state_S, \cdot, D(\cdot))}(state_V) = 1]
\]
is negligible in \(k\). (Immediately above, the notation \(\overline{\text{Ver}}_1^{\text{Sim}_3(state_S, \cdot, D(\cdot))}\) means that the adversary gets to choose \(x\) and will receive the result of \(\text{Sim}_3(state_S, x, D(x))\) – i.e., the two \(\cdot\) symbols refer to the same value).

Zero-knowledge is a useful property, but it may be too strong as a requirement. Instead, a weaker definition of semantic security could be used. Informally, semantic security is the property that the adversary doesn’t learn any more about some new message \(x'\) from existing proofs \(\pi_{x_1}, \pi_{x_2} \ldots\), than any other PPT adversary does from knowing only the pairs \((x_1, D(x_1)), (x_2, D(x_2)) \ldots\) The formal definition uses the semantic security of public-key encryption from [14] as a starting point.

**Definition 5.** The ADS is said to be \(\epsilon\)-semantically secure for a database \(D\) (where \(D'\) is a part of the database that is public knowledge) if, for every PPT adversary \(A\), there exists a PPT adversary \(A'\), such that

\[
\text{Pr}[f(x, D(x)) \leftarrow A(PK, D', Comm \overset{\$}{\leftarrow} \text{Auth}(PK, SK, D), Info = \{(x_1, \pi_{x_1}), \ldots (x_n, \pi_{x_n})\}) \text{ s.t. } (x, D(x)) \not\in Info \& x \in M_x(D)] \leq \\
\text{Pr}[f(x, D(x)) \leftarrow A'(PK, D', Info' = \{(x_1, D(x_1)), (x_2, D(x_2)), \ldots (x_n, D(x_n))\}) \text{ s.t. } (x, D(x)) \not\in Info' \& x \in M_x(D)] + \epsilon
\]

where \((PK, SK) \leftarrow \text{Gen}(1^k)\) and \(M_x(D) \subseteq DS\) is some space where \((x, Out)\) occurs with probability at most \(p \approx \frac{1}{\text{poly}(k)}\) for every \(x \in M_x(D)\), for all polynomially bounded functions \(f : \{0, 1\}^* \rightarrow \{0, 1\}^*\). The advantage of the adversary \(A\), i.e., \(\text{Adv}^{SS}_A(1^k, D) = \epsilon\).

The restriction in place is required, because a prediction adversary would otherwise select a random \(x\) from \(\{0, 1\}^*\) and guess \(Out\). The probability of success in that case
approaches 1, since the database itself is polynomial sized, and negligible in size compared to the domain \( \{0,1\}^* \). But obviously, such a randomly constructed string is of no practical significance to the scheme.

Note that, we can allow the adversaries to construct the sets Info and Info' adaptively by providing them oracle access to Res and the database \( D \) respectively.

For the semantic security for PKE in [14], the definition explicitly allows the adversary access to auxiliary information about the messages (e.g., the distribution of messages). We have avoided explicitly mentioning the same only for notational convenience, but we allow the adversary the knowledge to a public component of \( D \).

### 2.2 The DNSSEC standard and NSEC3

First, we present a simplified version of the DNSSEC (including NSEC3) using the definition of the DNSSEC/ADS model presented above. The scheme doesn’t describe the authentication or for that matter take into account DNSKEY distribution at all!

**Parameters** \( (1^k) \) For a security parameter \( k \), the underlying signature algorithm sign (for e.g., RSA, EC-DSA) and the hash-algorithm hash (for e.g., SHA-256, SHA-512) are selected. For the sake of analysis we consider hash to be a random oracle.

**Gen** \( (1^k) \) The Gen \( (1^k) \) runs the key-generation algorithm of one of the signature schemes used (for e.g., the RSA signature key generation algorithm). Let the output be \( (PK, SK) \). The same is authenticated using the authentication chain.

**Auth** \( (PK, SK, D) \) For the set of records \( D \) that the owner maintains, the algorithm does the following:

For each record \( R \), having domain name \( u \), compute \( h_u = \text{hash}(u) \) and signature \( \sigma_u = \text{sign}(u, D(u)) \) and store in CD (along with \( u \) and \( D(u) \), where \( D(u) \) is the corresponding information from the zone).

CD is sorted according to the values \( h_u \). Now, for every record, store also the value
$h_{u'}$ of the previous record in the list in CD. Also, a signature $\sigma_{u,u'}$ on $(h_u, h_{u'})$ is stored with the record. Henceforth, we call this part of the record, i.e., the tuple $(h_u, h_{u'}, \sigma_{u,u'})$, as the NSEC3 record for this tuple. Thus Auth outputs CD as the trapdoor. There is no Comm output here.

\textbf{Res} (CD, PK, x) For the query on some $x$, Res computes $h_x = \text{hash}(x)$ and searches CD for $h_x$. If $x$ exists in CD, then, Res assigns $\pi_x = (x, D(x), \sigma_x)$. If $x \not\in$ CD, then, it finds the smallest value $h_u \in$ CD such that $h_u > h_x$ and assigns to $\pi_x$ the corresponding NSEC3 record.

Res returns $\pi_x$.

\textbf{Ver} (PK, x, $\pi_x$) Ver runs the verification procedure corresponding to sign. Additionally, if $\pi_x$ is a NSEC3 record, then it verifies if $h_{u'} < h_{u''} < h_u$. If valid, then Ver returns $D(x)$ (if returned) or Out (if NSEC3). Otherwise, Ver returns Invalid.

While do not provide concrete security proofs for completeness and soundness, an intuition for why the above scheme is both complete and sound are:

\textbf{Completeness} Completeness comes from the fact that the hash function hash is a well-defined function and the correctness of the signature. If indeed the verifier rejects the signature, either the signature was on a value different for the one the verifier is using or the signing the algorithm is faulty. If Res and Ver used different values it means the hash(x) computed by both doesn’t match.

\textbf{Soundness} The soundness follows from the unforgeability of the signature scheme and the collision resistance of the hash function. If for some $x \not\in D$, suppose the malicious responder Res outputs a postive record with signature on $h_y$ (where $y \in D$), then Res has found a collision since hash$(x) = \text{hash}(y)$. If we don’t have a collision then $h_x$ is a fresh message. Therefore, we have a forgery! Similarly, if Res generates an NSEC3 response for $x \in D$, again, we will have a forgery.

\footnote{The NSEC3 records actually use a salt. This paper \cite{8} says that the use of salt in NSEC3 is ineffectual.}
2.3 Pruned Merkle Tree – Hybrid Scheme

We only briefly described the hybrid scheme here. The scheme is a plausible restriction of the scheme for zero-knowledge databases in [10], to allow only offline computations, i.e., proofs are just traversals on the precomputed Merkle Tree. Unlike in their protocol, we restrict the size of negative proofs (that are not in the Merkle Path) to only size $d$. Their scheme uses Mercurial Commitments, but we shall use the usual commitment schemes (COMMIT, OPEN, VEROPEN)\(^3\) as the malleability provided by soft-commitments isn’t required for us. Commitment schemes achieve the property of binding and hiding.

We do not provide formal definitions and refer readers to [24] for the same.

Another difference is that our scheme is ordered on the outputs of a random oracle, rather than on the actual inputs $x$. We do this to bring in some amount of unpredictability, i.e., if we query on $x$, it is trivial to query on it’s neighbour $x'$ where the last bit of $x$ is flipped. But if we order by the output of random oracle, then for our query on $x$, with $r_x = RO(x)$, querying the neighbour of $x$, i.e., the value $y$ for which the $RO(y) = r'_x$, where $r'_x$ is $r_x$ with its last bit flipped, is infeasible. We can succeed in querying the neighbour with probability $\frac{1}{2^k}$, where $k = |r_x|$.

Let us assume that the hybrid scheme uses a random oracle $RO$ with output length of $k$ bits. Let $PK_C$ be some public key of the commitment scheme and $SK_{sign}$ and $PK_{sign}$ the signing and verification key of the authenticator Auth. Therefore, the authenticator’s public key is $PK = (PK_C, PK_{sign})$. Let $D$ be the database (that the hybrid scheme aims to protect) of size $m$, containing pairs $(x, y)$ indexed on $x$. The scheme returns proofs of size $d$ for authenticated denial of existence (security level). Here, $d$ is small enough, that $2^d$ computations is feasible and preferably practical. The scheme (paraphrased from [10] along with the aforementioned changes) briefly entails the following steps:

**Commitment** The commitment to the database is the commitment at the root of an “incomplete” Merkle Tree. For all pairs $(x, v_x)$ in the database, query $RO$ on $x$ to

---

\(^3\)The definition of commitments in [24] don’t have the protocol OPEN, but they infact merge COMMIT and OPEN. We split the same for consistency with notation from Mercurial Commitments in [10]
receive $r_x \leftarrow \mathcal{RO}(x)$. Compute $c_x = \text{Commit}(PK_C, r_x, v)$. If the neighbour of $x$ (by the random oracle ordering) is not in the database, then $c_y = \text{Commit}(PK_C, r'_x, \perp)$ where $\perp$ is a special symbol signifying that the query isn’t in the database. For all other leaves $x$, $c_x = \text{nil}$. We now build a tree in bottom-up fashion from these defined leaves, where at every level $i$ from $k-1$ to 0, for each string $\sigma$ of length $i$, we compute $c_\sigma$ as follows:

1. If $c_{\sigma 1} \neq \text{nil}$ and $c_{\sigma 0} \neq \text{nil}$ then $c_\sigma = \text{Commit}(PK_C, \sigma, (c_{\sigma 1}, c_{\sigma 0}))$.

2. Else if either $i \leq d$ or $c_{\sigma'} \neq \text{nil}$ where $\sigma'$ is $\sigma$ with it’s last bit flipped , then $c_\sigma = \text{Commit}(PK_C, \sigma, \perp)$.

3. Else $c_\sigma = \text{nil}$.

The commitment at the root is $c_\emptyset$. If the database is empty, then the root has $c_\emptyset = \text{Commit}(PK_C, \emptyset, \perp)$. Auth outputs the Merkle tree and the database $D$ as the committed database $CD$. But we overload the use of $CD$ here to refer only to the Merkle tree, and we refer to $D$ separately. It outputs $\text{sign}(c_\emptyset)$ as $Comm$.

**Query Responses** The responder $Res$ now has $CD$, $D$ and $Comm$ with it. To answer a query $x$ which is in the database, we use the same method as in [10]. We elaborate the same verbatim from [10]. Let $r_{x|i}$ be the first $i$ bits of $r_x$, and $(r_{x|i})'$ be the first $i-1$ bits of $r_x$ followed by the $i$-th bit flipped. Let $\varpi_x = \text{OPEN}(PK_C, D(x), r_x, c_x)$ and $\varpi_{r_{x|i}} = \text{OPEN}(PK_C, (c_{r_{x|i}0}, c_{r_{x|i}1}), r_{x|i}, c_{r_{x|i}})$ for $0 \leq i < k$. Return $D(x)$ together with $c_{r_{x|i}}, c_{(r_{x|i})'}$ for $1 \leq i \leq k$ and $\pi_{r_{x|i}}$ for $0 \leq i \leq k$. In other words, $Res$ returns $Comm$, $D(x)$ together with the proof $\pi_x$ being its authenticating path to the root, which consists of ancestors of $x$, their siblings, and proofs that each parent is the commitment to the two children.

Suppose, $x$ is not in the database. We repeat the same process as above, but only for levels $0 \leq i \leq t$, where level $t$ is the one where $\varpi_{r_{x|i}} = \text{OPEN}(PK_C, \perp, r_{x|t}, c_{r_{x|i}})$. Return $Comm$, $D(x)$ together with $c_{r_{x|i}}, c_{(r_{x|i})'}$ for $1 \leq i \leq t$ and $\varpi_{r_{x|i}}$ for $0 \leq i \leq t$. 
The proof $\pi_x$ here is the authenticating path, but only till level $l$. Note, that $t \geq d$.

**Verification** For verification $\text{Ver}$ first verifies the commitment is indeed from $\text{Auth}$ by $\text{Ver}_{\text{sign}} (PK_{\text{sign}}, \text{Comm})$. Then it runs $\text{VEROPEN}$ on all the returned proofs (Merkle tree path verification). If at least one of the proofs doesn’t verify, it outputs $\text{Invalid}$. If one of the proofs opened to $\bot$, then it accepts that $x$ isn’t in the database, else, if there are $k$ proofs and the final proof on $r_x$ opens to $v$, it accepts that $D(x) = v$.

Again, we do not provide formal arguments for the completeness and soundness of this protocol. The two must follow from the completeness and soundness of the corresponding zero-knowledge protocol in [10]. An intuitive argument for the same does follow.

**Completeness** Completeness of the hybrid scheme follows from the completeness of the underlying commitment scheme used, the correctness of the signature, key distribution and the consistency in the output of the random oracle. Suppose a $\pi_x$ generated by $\text{Res}$ isn’t validated by $\text{Ver}$, then either both have differing copies of $r_x$ or $PK$. If the signature validations doesn’t fail, then one node in the authentication path fails verification going against completeness of the underlying commitment scheme.

**Soundness** Soundness of the hybrid scheme follows from the binding of the commitment scheme and the existential unforgeability of the signature. $\text{Res}$ cannot generate its own Merkle tree, because it will then have to output the right $\text{Comm}$, thus forging a signature for the public key $PK$. If it uses the same Merkle root, then one of the internal nodes commits to at least two differing values, thus breaking the binding property of the commitment scheme.

### 2.4 Signatures from DNSSEC

In this section, we investigate the possibility of construction of secure signatures using a secure DNSSEC scheme. We reduce the unforgeability of the signature to the soundness and the semantic security of the ADS.
2.4.1 Signature Definitions

For our purposes we require a restricted version of existential unforgeability. This property is defined formally.

**Definition 6.** A signature scheme \((\text{Gen}_{\text{sign}}, \text{Sign}_{\text{sign}}, \text{Ver}_{\text{sign}})\) has restricted existential unforgeability under chosen message attacks if every PPT forger \(\mathcal{FS}\) has a success probability that is negligible in the security parameter, in the following game:

1. \(\mathcal{FS}\) receives the public key \(PK_{\text{sign}}\) and other public parameters (obtained by running \text{Setup} and \text{Gen}_{\text{sign}} for security parameter \(k\)). Also, it receives a target message space \(\mathcal{M}_x\).

2. \(\mathcal{FS}\) is allowed to make at most some \(\text{poly}(k)\) queries to the signing oracle \(S\).

3. \(\mathcal{FS}\) wins the game if it outputs a valid forgery \(\sigma_x\) (i.e., \(\text{Valid} \leftarrow \text{Ver}_{\text{sign}} (PK_{\text{sign}}, \sigma_x, x)\)) on some \(x \in \mathcal{M}_x\) such that it had not queried on \(x\) to the signing oracle \(S\).

This definition doesn’t specify the extent of restriction. The extent of restriction is decided by the size of \(\mathcal{M}_x\). In case of an exponentially large (in \(k\)) \(\mathcal{M}_x\), the forger should be given a polynomial sized description of the entire space (for e.g., the basis of a linear space).

2.4.2 Construction

The construction of a signature scheme \((\text{Setup}, \text{Gen}_{\text{sign}}, \text{Sign}_{\text{sign}}, \text{Ver}_{\text{sign}})\) using a secure ADS \((\text{Gen}, \text{Auth}, \text{Res}, \text{Ver})\) is as follows:

\text{Setup}(1^k) \quad \text{Given the security parameter, selects a (known/public) database } D' \text{ of size } n' \approx \text{poly}(k).

\text{Gen}_{\text{sign}}(1^k, D') \quad \text{Given the security parameter as input the key generation process proceeds in the following manner.}

- \((PK, SK) \leftarrow \text{Gen}(1^k)\)
• Select $\mathcal{M} \subset DS \times \{0,1\}^*$ be a random space of pairs $(x,y)$ where $y \neq Out$, s.t. $|\mathcal{M}| \approx poly(k)$. Let $\mathcal{M}_x$ be the space consisting of all the $x$ values in $\mathcal{M}$.

• $D$ is the database obtained by appending to $D'$ pairs from $\mathcal{M}$, such that every pair in $\mathcal{M}$ is included with probability $\frac{1}{2}$.

• $(Comm, CD) \leftarrow Auth(PK, SK, D)$

The signing key is $SK_{sign} = CD$ and the verification key is $PK_{sign} = (PK, Comm)$. $\text{Sign}_{sign}(SK_{sign}, PK_{sign}, m)$ The signature $\sigma_m$ is extracted by running Res, i.e., $\pi_m \leftarrow \text{Res}(CD, PK, m)$. The signature $\sigma_m = \pi_m$.

$\text{Ver}_{sign}(PK_{sign}, m, \sigma_m)$ The signature verification algorithm parses $PK_{sign}$ to obtain $PK$ and $Comm$. Then, it runs the verification algorithm $\text{Ver}$ to validate the signature, i.e., $y \leftarrow \text{Ver}(m, \sigma_m, PK, Comm)$. Output $\text{Invalid}$ if $y = \text{Invalid}$, $\text{Valid}$ otherwise.

2.4.3 Security Proof

In our construction, to forge the signature on some message $x$ the forger would require to guess the existence of the message in $D$ (and possibly the appropriate value $D(x)$). Thus, during a forgery, we expect to encounter one of the two cases:

1. For the message $x$, $\mathcal{F}_S^S$ can produce a forgery by predicting incorrectly the value of $D(x)$, i.e., for an arbitrary pair $(x,y)$ where $y \neq D(x)$.

2. For the message $x$, $\mathcal{F}_S^S$ can produce a forgery by correctly predicting the value of $D(x)$.

Firstly, we would like to show that no PPT forger $\mathcal{F}_1^S$ on the target $\mathcal{M}_x$ (as per definition 6) can produce a forgery by “lying”, i.e., forging in case 1 is infeasible. We show that if the forger wins in the first case, then we can use such a forger to break the soundness of the underlying ADS.
Lemma 2.4.1. If there exists a forger $F_{1}^{S}(PK_{\text{sign}}, D', \{0,1\}^{\ast})$ running in time $\tau$, that can forge using some pair $(x, y)$ where $y \neq D(x)$ with non-negligible probability $\epsilon$, then there exists an adversarial host, $\overline{\text{Res}}$ running in time $\tau$ that breaks the soundness property of ADS with probability $\epsilon$.

Proof. $\overline{\text{Res}}$ on receiving $(\text{Comm}, \text{CD}, PK)$ as input, uses the forger $F_{1}^{S}$ to break soundness in the following manner.

- $\overline{\text{Res}}$ selects a random subset $D' \subset D$ and sets target space as $\{0,1\}^{\ast}$.
- $\overline{\text{Res}}$ sets $PK_{\text{sign}} = (PK, \text{Comm})$ and runs $F_{1}^{S}$ with input $(PK_{\text{sign}}, D', \{0,1\}^{\ast})$.
- $\overline{\text{Res}}$ simulates the signing oracle $S$’s response to a query on some $x_{i}$ by $F_{1}^{S}$, by running $\text{Res} (\text{Comm}, \text{CD}, PK)$ and returning its output $\pi_{x_{i}}$ as the signature $\sigma_{x_{i}}$. The query and response are stored as state information.
- On receiving the forgery $\sigma$ on some message $x$ from $F_{1}^{S}$, $\overline{\text{Res}}$ computes $y \leftarrow \text{Ver}(x, \sigma, \text{Comm}, PK)$. If $y = \text{Invalid}$ or $y = D(x)$ or $x$ is an earlier query to the signing oracle, it aborts citing failure, else, it returns $(x, y)$ and terminates.

Since, we assume in this lemma $F_{1}^{S}$ produces a forgery using a pair $(x, y)$, where $y \neq D(x)$, $\overline{\text{Res}}$ wins the soundness game whenever the forger wins the game. Therefore, it can win the soundness game with probability $\epsilon$. It is seen that the running time is $\tau$.

Now that we have reduced the possibility of a “lying” forger to soundness, we show a reduction of a weaker forger $F_{2}^{S}$ to semantic security. The forger is restricted in the sense that, it must output a forgery on some $x \in M_{x}$ for which it hasn’t already queried the signing oracle. It is seen that with access to only $D$ as an oracle, even with the best prediction adversary (on $M_{x}$) $A'$ wins with probability at most $\frac{1}{2}$.

Theorem 2.4.2. Suppose, there exists a PPT forger $F_{2}^{S}$ running in time $\tau$, that forges successfully on some value $x$ in $M_{x}$ with probability $\epsilon$, then, there exists an adaptive privacy adversary $A$ against the ADS (with soundness guarantee) s.t. $\text{Adv}_{A}^{SS}(1^{k}, D) \approx \epsilon$. 

Proof. The adaptive (stateful) privacy adversary $A^{\text{Res}}(PK, D', \text{Comm})$ with oracle access to Res, uses $F^S_2$ to output some $(x, D(x))$ such that $x \in M_x$. $A^{\text{Res}}$ is constructed in the following manner.

1. $A$ runs $F^S_2$ on the input $(D', PK_{\text{sign}}, M_x)$, where $PK_{\text{sign}} = (PK, \text{Comm})$.

2. To respond (by simulating the signing oracle $S$) to a query on any message $x_i$ by $F^S_2$, $A$ queries the same $x_i$ to Res and returns the received proof $\pi_{x_i}$ as the signature $\sigma_{x_i}$. It stores all the queries and the responses as state information.

3. Suppose, $F^S_2$ outputs a forgery $(x, \sigma_x)$ where $x$ was never queried to the signing oracle. If $x \in M_x$ and $y \leftarrow \text{Ver}(x, \sigma_x, PK, \text{Comm})$ where $y \neq \text{Invalid}$, then $A$ outputs $(m, y)$ and terminates. Otherwise, if $y = \text{Invalid}$, $A$ outputs $(x, \text{Out})$, else, $A$ selects a $x' \in M_x$ which hasn’t queried to Res earlier, and outputs $(x', \text{Out})$ and terminates.

It can be seen that $A$ runs in time $\tau$, i.e., the time taken by $F^S_2$. Also, $A$ succeeds whenever $F^S_2$ succeeds, except if $F^S_2$ lied about the value of $y$. We see from lemma 2.4.1 that if $F^S_2$ succeeds with $y \neq D(x)$, then we can break the soundness of the ADS. Since, the ADS comes with soundness guarantees, we assume, this happens with a probability negligible in $k$, i.e., $\negl(k)$. When $F^S_2$ fails, then $A$ wins with probability $\frac{1}{2}$, given the construction of $D$. Therefore, $\text{Adv}^{\text{SS}}_A(1^k) \leq \epsilon - \negl(k) \approx \epsilon$. \qed

2.4.4 Offline?

We were able to achieve successful reduction to signatures with very weak unforgeability guarantees. But the above reduction isn’t helpful to claim that secure offline-only ADS is impossible. The reason is that existence of offline signatures with aforementioned guarantees isn’t surprising. One could pre-compute strong signatures on the all $x \in M_x$ and for all $x \notin M_x$ simply return $x$ itself.

The reason we cannot reduce to conventional EF-CMA signatures, is that ADS doesn’t hide the non-existence of arbitrary domains. It is in-fact impossible to hide the non-
existence of arbitrary domains in the database. This is so because the size of the database as compared to the size of the domain space DS is negligibly small.
Chapter 3

Defining Zone Enumeration

In the previous section, we saw how a conventional notion of security couldn’t capture our desired impossibility result. We now shift our approach. Instead of proving through reductions, we would like to show theoretical zone-enumeration attacks on all plausible ADS mechanisms that adhere to DNSSEC’s operational constraints.

In DNSSEC, privacy is concerned with learning all/most elements within a zone, not learning elements that are not in the zone. We present two styles of adversaries Oblivious and Adaptive (rather notions of “insecurity”) to capture this scenario. Further, we show these attacks in the known scenario of NSEC3 and the example hybrid proof system.

3.1 Defining Privacy

We look at defining privacy, rather, the lack of privacy, through some attacks. The attacks defined here assume that attackers (at some point in time) get a dictionary $\text{Dict}$ of size $n$. The attackers are given oracle access to $\text{Res} (\text{CD}, \text{PK}, x)$ for some database $D$ of size $m$. We currently work with the restriction that $D \subset \text{Dict}$. An adversary is then expected to “prune” $\text{Dict}$, with minimum number of queries to $\text{Res}$, to better predict the database $D$. We make no assumptions on the computational capabilities of the attackers, except that it is bound to some polynomial in the size of the database. We only attempt to minimize the number of queries made to $\text{Res}$.

Note that we make a very strong assumption on the nature of the dictionary. We can avoid such a requirement. But, then the adversary is expected to predict only $D \cap \text{Dict}$. 
Now, with the scenario established, the attacks are defined below. The distinction between the two attacks is on the use of the dictionary in querying Res. Also, the definition will concretely establish what is expected of a “pruned” dictionary.

**Oblivious Attack** An \((\delta, \varepsilon, n, m)\)-oblivious attacker (having \(\text{Dict}\) as input and oracle access to \(\text{Res}\)) has two algorithms.

\[
\text{Learner}^{\text{Res}}(\delta, \varepsilon, n) \rightarrow \text{state}_L
\]

The learner is allowed to query \(\text{Res}\) multiple times, before it outputs some state, \(\text{state}_L\). The learner essentially learns the function \(\text{Res}\). The state, \(\text{state}_L\) can be thought of as a description of the learned function. Note that the learner is given the size of the dictionary \(n\) and success parameters \(\varepsilon\) and \(\delta\), but is not given the size of the database \(m\) and doesn’t have access to the dictionary.

\[
\text{Prune}(\delta, \varepsilon, \text{Dict}, \text{state}_L) \rightarrow \text{pDict}
\]

The pruning adversary gets the learned algorithm through \(\text{state}_L\). The pruning algorithm prunes the dictionary to output \(\text{pDict}\). Again, the pruning algorithm isn’t given the size of the database \(m\) and oracle access to \(\text{Res}\).

The oblivious attacker produces \(\text{pDict}\) as its output. An oblivious attacker is a \((\delta, \varepsilon, n, m)\)-oblivious attacker, if it outputs some \(\text{pDict}\), such that

\[
\Pr[x \overset{R}{\leftarrow} \text{pDict} : x \in D] \geq \varepsilon
\]

with probability \(\geq \delta\) over all the choices of \(D\) of size \(m\) and \(\text{Dict}\) of size \(n\) (where \(D \subseteq \text{Dict}\)).

**Adaptive Attack** An adaptive attacker is given a dictionary \(\text{Dict}\) and access to \(\text{Res}\). An adaptive attacker outputs a pruned dictionary \(\text{pDict}\). The adaptive attacker is a \((\delta, \varepsilon, n, m)\)-Adaptive attacker, if it outputs some \(\text{pDict}\) using \(o(n)\) queries to \(\text{Res}\), such that

\[
\Pr[x \overset{R}{\leftarrow} \text{pDict} \land x \in D] \geq \varepsilon
\]
with probability $\geq \delta$ over all the choices of $D$ of size $m$ and $\text{Dict}$ of size $n$ (where $D \subseteq \text{Dict}$).

The adaptive attacker unlike the oblivious attacker is allowed to query $\text{Res}$ using the dictionary. Note that if we allow $O(n)$ queries, then all schemes are vulnerable to an adaptive attack. An adaptive attacker will simply exhaust the dictionary by querying the same to $\text{Res}$.

In our definitions, the adversary considers all elements in the dictionary are equally likely to be in the database. In other words, the adversary doesn’t have additional auxiliary information in its possession. Furthermore, in the oblivious attack we make an additional generalisation. The distribution of proofs on elements in the dictionary that are not in the database must approximate the distribution of proofs on elements in the whole domain space $\text{DS}$. This is expected from any reasonable protocol. To elaborate, for any reasonably selected dictionary $\text{Dict}$, the distribution of proofs on the remaining $n - m$ elements in $\text{Dict}$ must approximate the distribution of proofs over the entire domain space $\text{DS}$. If this weren’t the case, it would imply that a biased dictionary would be easy to construct, that would imply that the proofs are trivially predictable. Such a protocol is trivially not private. The above restriction is realised if the proofs are computed on $\text{hash}(x)$ instead of $x$ itself.

The above definitions are loosely based on PAC learners with membership queries [33].

**Note:** *Feasibility of Oblivious Attacks* - Following this section, we try to construct only oblivious adversaries. Clearly oblivious adversaries are stronger than their adaptive counter parts as they have to learn the proofs without access to the dictionary. Further, they may also be more practical. In a typical practical scenario, an oblivious attack can be mounted against a server at any given time, by simply anticipating that we will get a reasonable dictionary at some point in the future. Once, we get a dictionary, we can learn the elements in the database at the time of attack, *i.e.*, it is resilient to updates in the
database. In case of a rate-limiting server, we need not flood the server with such queries. We can spread the queries evenly over the period where we are certain that the zone isn’t updated and the keys aren’t rolled over. The learner could also sample already queried packets from the wild, instead of actually querying \texttt{Res} itself.

### 3.2 Oblivious Attacks

We shall now describe oblivious attacks on certain example \texttt{Res} functions. The first \texttt{Res} function is the NSEC3 and the next is an example construction from a pruned-offline version of a merkle-tree based construction of zero-knowledge databases.

#### 3.2.1 NSEC3

In the following oblivious attack on NSEC3, we consider that the security parameter $\kappa$ is some adversarial estimate of the size of zone/database. For details on NSEC3 we refer the reader to section 2.2.

Our attack on NSEC3 works under the assumption that the scheme uses a random oracle instead of a conventional cryptographic hash function. This assumption helps simplify the probability analysis.

Prior to proceeding, note that NSEC3 uses adjacent random oracle outputs $(h_1, h_2)$ as a proof bucket. We would like to bound the bucket sizes. As a first step we make use of the result of finding near birthdays from [4]. The bound for the same follows. It is written verbatim from [11, Section 2.3] (with minor change in notations), but as a lemma here:

**Lemma 3.2.1.** Let $p(\kappa, m, t)$ denote the probability that in a group of $m$ people, no pair\footnote{The statement in [11] has a typo. It gives the above inequality saying at least one pair exists, but it is actually the case that no pair exists.} with birthdays within $t$ days of each others exists, if there are $\kappa$ equally likely birthdays, then

$$p(\kappa, m, t) = \frac{(\kappa - mt - 1)!}{\kappa^{m-1}(\kappa - m(t + 1))!}$$
Thus, in our case the likely number of birthdays is the range of the random oracle (of size $2^k$, say). Therefore, $\kappa = 2^k$. Also, the number of people will be the number of elements in the database, i.e., $m$. We need to identify $t$ in this case. We could set $p(\kappa, m, t) = \epsilon$ according to some adversarial confidence requirement (i.e., with confidence $1 - \epsilon$ we can estimate the size of $t$). We can estimate $t$ from $2^k$, $m$ and $\epsilon$ then. But there is a caveat! The above birthday problems the distance $k$ is cyclical, i.e., birthday at the end of the year and a birthday at the start of the next year should also be $t$ calendar days apart. We do not require the same in our case. Thus, we simplify the proofs to neglect the wraparounds in [4] to identify the right probability.

**Lemma 3.2.2.** If $p'(\kappa, m, t)$ is the probability that in a group of $m$ people, no pair have a birthday within $t$ days of each other, in the same year, for a year with $\kappa$ days, then

$$
p'(\kappa, m, t) = \frac{(\kappa - (m - 1)(t - 1))!}{\kappa^m(\kappa - (t - 1)(m - 1) - m)!}\]

Proof. Following the lines of proof from [4] : Consider the set of all possible sequences $1 \leq x_1 \leq \ldots \leq x_m \leq \kappa$, such that $t \leq |x_{i+1} - x_i|$ for $i \leq m - 1$. Then, our required event consists of all possible permutations ($m!$) of each of such sequences (since, the numbers are chosen uniformly at random and we are interested only in the distance between the adjacent pairs in sorted order of these sequence of numbers).

For every such sequence $1 \leq x_1 \leq \ldots \leq x_m \leq \kappa$, such that $t \leq |x_{i+1} - x_i|$ for $i \leq m - 1$, there exists a sequence $1 \leq y_1 \leq \ldots \leq y_m \leq \kappa - (m - 1)(t - 1)$, where $y_i = x_i - (i - 1)(t - 1)$ for all $1 \leq i \leq m$. Therefore, we now have a sequence where $y_{i+1} - y_i \geq 1$. Therefore, the total such choices of $y$’s are $\binom{\kappa-(m-1)(t-1)}{m}$.

Hence, total possible sequences are $m!\binom{\kappa-(m-1)(t-1)}{m}$. Thus,

$$
p'(\kappa, m, t) = \frac{m!\binom{\kappa-(m-1)(t-1)}{m}}{\kappa^m(\kappa - (m - 1)(t - 1) - m)!}$$
Note that for $n > m$, it is imminent that $p'(\kappa, n, t) < p'(\kappa, m, t)$. Now, we can use lemma 3.2.2 to estimate $t$, given $n$ (the size of $\text{Dict}$, with the knowledge that $n > m$), estimated confidence interval $\epsilon$, and $\kappa = 2^k$. We see that

$$
\epsilon = \frac{(2^k - (n-1)(t-1))!}{2^{nk}(2^k - (n-1)(t-1) - n)!} > \frac{(2^k - (n-1)(t-1) - (n-1))^n}{2^{nk}} = (1 - \frac{(n-1)t}{2^k})^n
$$

Therefore,

$$
\epsilon^{1/n} > (1 - \frac{(n-1)t}{2^k})
$$

Similarly,

$$
\epsilon < \frac{(2^k - (n-1)(t-1))^n}{2^{nk}} = (1 - \frac{(n-1)(t-1)}{2^k})^n
$$

Giving us,

$$
t > \frac{(1 - \epsilon^{1/n})2^k}{(n-1)}
$$

$$
t < \frac{(1 - \epsilon^{1/n})2^k}{(n-1)} + 1
$$

Therefore, for simplicity, we consider $t \approx \frac{(1 - \epsilon^{1/n})2^k}{(n-1)}$. This implies that NSEC3 leaves buckets of size at least $t$, with probability at least $\epsilon$ (note that the above calculations were based on the size of $\text{Dict}$ and not the database and the database must in fact be smaller). The probability that an element falls in such a bucket is $\frac{(1 - \epsilon^{1/n})2^k}{(n-1)} = \frac{(1 - \epsilon^{1/n})}{(n-1)}$.

Armed, with the above bounds, we shall now present an attack on zone-enumeration. Here we present a learner that takes as input $\epsilon$ and $n$, and not $(\delta, \epsilon, n)$. We shall later compute $\delta$ and $\epsilon$ from these two values.
Learner\textsuperscript{Res}(\epsilon, n) The learner in this case outputs \(\mathbb{CD}\) as its state \(L\). It works as follows:

1. Initially, \(\mathbb{CD}\) is empty.
2. Generate \(x_{\text{new}} \overset{R}{\leftarrow} \mathcal{DS}\) such that its proof isn’t in \(\mathbb{CD}\). In other words, \(\mathbb{CD}\) shouldn’t contain the NSEC3 pair \((h_1, h_2)\), such that \(\mathcal{RO}(x_{\text{new}}) \in [h_1, h_2]\). If the learner cannot find such an \(x_{\text{new}}\) in \(n^2(1-\epsilon/n)\) trials, it returns \(\mathbb{CD}\) and terminates.
3. Query \(\text{Res}\) on \(x_{\text{new}}\) to receive proof \(\pi_{\text{new}}\).
4. If \(\pi_{\text{new}}\) is valid, add to \(\mathbb{CD}\). Go to step 2.

Prune(\(\mathbb{Dict}, \mathbb{CD}\)) For every element \(y\) in \(\mathbb{Dict}\), Prune does the following:

1. Compute \(h_y := \mathcal{RO}(y)\).
2. In \(\mathbb{CD}\) if there exists some \(H_i = (h_{1i}, h_{2i})\) where \(h_y = h_{1i}\) and some \(H_j = (h_{1j}, h_{2j})\) where \(h_y = h_{2j}\), then add \(y\) to \(p\mathbb{Dict}\).
3. If \(\mathbb{CD}\) doesn’t contain any pair \(H_i = (h_{1i}, h_{2i})\) such that \(h_y \in [h_{1i}, h_{2i}]\), then add \(y\) to \(p\mathbb{Dict}\).

We shall now describe the correctness and efficiency of the above attack.

**Theorem 3.2.3.** Given the confidence factor \(\epsilon\), the described oblivious adversary is a \((\delta, \epsilon, n, m)\)-oblivious attacker, making at most \(m\) queries to \(\text{Res}\) in the learning phase, over all choices of databases \(D\) of size \(m\) and \(\mathbb{Dict}\) of size \(n\), such that \(D \subseteq \mathbb{Dict}\), \(\delta \approx 1 - (\frac{m}{e^2} + \frac{m^2}{2e})\)
and \(\epsilon \approx \frac{m}{n(1-\epsilon^2/n) + me^{1/n}}\).

**Proof.** Firstly, for correctness, the pruner requires to get all possible buckets of size \(t\).
We prove that the above learner doesn’t learn a NSEC3 bucket of size \(t\) with negligible probability. Note that the probability that the bucket is not learned is at most \(p = \left(1 - \frac{1-\epsilon^2/n}{(n-1)}\right)^{n^2/(1-\epsilon^2/n)}\), i.e., the probability of terminating without being able to retrieve the said bucket. We see that \(p < \left(1 - \frac{1-\epsilon^2/n}{(n-1)}\right)^{(n-1)n/(1-\epsilon^2/n)}\), viz. \(p < \frac{1}{e} \cdot \frac{n}{e^n}\). Therefore, since there are at most \(m\) such \(t\) sized buckets, the total probability of failure \(1 - \delta < \frac{m}{e^m}\).
Therefore, \( \delta \approx 1 - \frac{m}{en} + \frac{m^2}{2e} \) (since \( \delta \) by itself is a lower-bound, we do away with the inequality).

Next, we show that the learner makes at most \( m \) queries to learn these buckets. By construction, the learner makes a new query only when it generates \( x_{\text{new}} \). In other words, every query adds a new bucket to \( CD \), unless the query collides with a random oracle value. Both of the above happen with probability \( \approx \frac{m^2}{2e} \). Therefore, \( \delta \geq 1 - \frac{m}{en} + \frac{m^2}{2e} \). Also, the number of queries made is exactly \( |CD| \), which is \( \leq |CD| \approx m \). Therefore, the learner makes at most \( m \) queries to \( \text{Res} \).

Given \( CD \) is sufficient for the pruner to enumerate all elements in the database. Let us consider some \( x \in D \). If the proof is already in \( CD \), then we are done. Else, if \( RO(x) \) is among the boundaries of \( x \in D \), again by construction we are done. Else, the bucket for which the same occurs as a boundary, has size much less than \( t \). By construction, any element in these said smaller buckets are already included in \( \text{pDict} \). Therefore, all \( m \) elements from \( D \) will be included in \( \text{pDict} \).

All additional elements in \( \text{pDict} \) must be from these smaller buckets. Note that \( CD \) must contain at least 1 bucket of size greater than \( t \) (since \( m << 2^k \)). Therefore, there are at most \((m - 1)\) buckets of size less than \( t \). Therefore the fraction of the dictionary that lies in this segment is thus, at most \( \frac{(n-m)(m-1)(1 - \epsilon^{1/n})}{n-1} < (n-m)(1 - \epsilon^{1/n}) \). Therefore, the expected size of the pruned dictionary \( |\text{pDict}| \approx m + (n-m)(1 - \epsilon^{1/n}) = n(1 - \epsilon^{1/n}) + me^{1/n} \). Thus, \( \epsilon \approx \frac{m}{n(1 - \epsilon^{1/n}) + me^{1/n}} \) (since \( \epsilon \) is a lower bound, we did away with the ‘\( > \)’ inequality).

3.2.2 Hybrid Scheme

The idea here is that we aren’t breaking the cryptographic structure of the scheme, but presenting an Oblivious attacker whose Learner exploits the combinatorial properties of the “restricted Merkle tree”. The scheme is presented in 2.3 and is a plausible approach to delegate all computations in [10] offline.

We define a bucket to be the set of \( 2^k - d \) leaves starting from values \( r_l \) to \( r_u \). The \( i \)-th bucket has \( r_l = i2^k - d \) and \( r_u = (i + 1)2^k - d - 1 \). Hence, the bucket \( i \) is the set of all leaves
under the $i$-th path in the tree of length $d$.

Let $D$ be the database (that the hybrid scheme aims to protect) of size $m$, containing pairs $(x, y)$ indexed on $x$. The scheme returns proofs of size $d$ for authenticated denial of existence (security level). Here, $d$ is small enough, that $2^d$ computations is feasible and preferably practical.

Let $n$ be the size of the dictionary that the adversary may obtain sometime in the future. Let $d'$ be the “confidence level” that the adversaries aim to achieve on each positive proof bucket that it retrieves. Let $c$ be a constant, denoting the maximum number of database elements that can lie within a bucket.

**Learner** The learner of the oblivious attacker $\text{Learner}^{\text{Res}}$ maintains the set of retrieved proofs $\Pi$. We overload the $\in$ operator here. For example, we use it both in the sense of $\pi \in \Pi$ for a proof $\pi$ and $r_x \in \Pi$ to denote that the set holds a proof for the value $r_x := RO(x)$. $\text{Learner}^{\text{Res}}$ maintains a set $\text{Yes}_{\text{Instance}}$ where he stores the set of proofs of length $d'$ or greater such that one of its leaf nodes is a database element. $\text{Learner}^{\text{Res}}$ begins by running the procedure $\text{Find}_{\text{Buckets}}$.

**Procedure** $\text{Find}_{\text{Buckets}}$

1. If $\Pi$ has $2^d$ distinct elements, terminate. Else, pick at random some message $x \xleftarrow{R} \text{DS}$ and $r_x = RO(x)$ such that $r_x \not\in \Pi$.

2. A query on $x$ returns a proof $\pi_x$. If the length of the proof is $d$, then add $\pi_x$ to $\Pi$ and go to step 1.

3. Given a proof $\pi_x$ of length greater than $d$. Let $r_x = RO(x)$ lie in the bucket $[r_l, r_u]$. Firstly, we store $\pi'_x$ in $\Pi$, where $\pi'_x$ is the first $d$ elements in the proof. Then, we call $\text{Interesting}_{\text{Region}}(r_l, r_u, (x, \pi_x))$. Go to step 1.

To elaborate on step 3, let the authenticating path have proofs $\varpi_{r_x|_i}$ for $0 \leq i \leq t$. Note that since, $\varpi_{r_x|_i}$ opens to $(PK_{C}, \bot, r_x|_t, c_{r_x|_t})$, by our construction, $c_{r_x|_t} \neq \text{nil}$,
where \((r_x|t)\)' is \(r_x|t\) with the last bit swapped. Therefore, according to the construction \((r_x|t)\)' is an internal node on an authenticating path to some element in the database. Therefore, the \(2^{k-d}\) bucket covering values from \(r_l\) to \(r_u\) is a bucket of interest.

**Procedure** \texttt{Interesting\_Region}((\(r_l\), \(r_u\), \((x, \pi_x)\))

Let \(\pi_x\) signify the proof region covered by the sibling of the last node of proof in \(\pi_x\).

1. If the length of \(\pi_x\) is greater than \(d'\), then let \([r_{l l'}, r_{u l'}]\) be the region of leaves covered under the subtree represented by \(\pi_x\). Store \([r_{l d'}, r_{u d'}]\) in \texttt{Yes\_Instance} and jump to step 4.

2. Select a random \(x' \xleftarrow{R} \text{DS}\) with \(r_{x'} := \mathcal{R}O(x')\) such that, \(r_{x'} \in \pi_x\).

3. Query on \(x'\) to obtain \(\pi_{x'}\). Let the length of \(\pi_{x'}\) be \(d_{x'}\). If \(d_{x'}\) is greater than \(d'\), store \(\pi_{x'}\) in the set \texttt{Yes\_Instance} and go to step 4. Otherwise, set \(\pi_x = \pi_{x'}\) and go to step 2.

4. Now consider \(\pi_{x}|-1\), \(\pi'|-1\), i.e., the proof region covered by the sibling of the node above the last one in \(\pi_{x}\). Now, search this region (the leaves under this subtree) again for longer proofs (indicating a smaller region containing another element in the database) i.e., if a randomly drawn element has proof size greater than \(d_{x'} - 1\), then search for a proof of size larger than \(d'\) and add the region \([r_{l x'}, r_{u x'}]\) covered by its sibling to \texttt{Yes\_Instance}.

5. Repeat step 4 by iteratively reducing the target proof size, \(\pi_{x}|-2\) and so on until the required proof size is down to \(d + 1\) (or until the space \(r_u \) to \(r_l\) is exhausted).

The procedure \texttt{Interesting\_Region} is in essence a depth-first search (DFS) through the subtree in the “interesting region” to find all regions of size \(\leq 2^{k-d'}\) elements which contain elements in the database.
The learner outputs \texttt{Yes\_Instance} as its state, \texttt{state}.

**Pruner** For every element \( x \) in the \texttt{Dict} with \( r_x = RO(x) \), if \( r_x \) falls in a region \([r_{lx}, r_{ux}]\), where \([r_{lx}, r_{ux}] \in \texttt{Yes\_Instance} \), then add \( x \) to \texttt{pDict}.

We shall now proceed to analyse the efficiency and the efficacy of the above oblivious attack. Now suppose an interesting region has \( c \) elements in the database. Since, we are using a random oracle, we can conjecture that the output of the random oracles corresponding to the above \( c \) elements must be evenly distributed between \( r_l \) and \( r_u \) too. We work with this assumption for the sake of a simple analysis. Therefore, the proofs in this interesting region are of length at least \( d + \lfloor \log c \rfloor \). Using this as a fact, we prove the following lemma.

**Note:** We don’t need to pre-determine \( c \). The constant is simply useful in the analysis. In effect we just require a DFS like search down the interesting region.

**Lemma 3.2.4.** The expected number of queries to generate a new element for the \texttt{Yes\_Instance} is \( 1 + (d' - (d + \lfloor \log c \rfloor))/2 \).

**Proof.** We define \( X_0 \) as the random variable signifying the number of queries made to generate a new element of the \texttt{Yes\_Instance} given the prefix of length \( d + \lfloor \log c \rfloor \). Let \( X_1 \) be the random variable signifying the number of queries made to generate a new element of the \texttt{Yes\_Instance} given a proof of length \( d + \lfloor \log c \rfloor + 1 \). We can define \( X_2, X_3 \ldots X_{d'-(d+\lfloor \log c \rfloor)} \) analogously. Note that, we are required to determine \( E[X_0] \). The above set of random variables forms a Markov Chain, the transition matrix of which is:

\[
P =\begin{bmatrix}
  d_0 & d_1 & d_2 & \ldots & d_{d'-(d+\lfloor \log c \rfloor)} \\
  d_0 & 0 & \frac{1}{2} & \frac{1}{2^2} & \ldots & \frac{1}{2^{d'-(d+\lfloor \log c \rfloor)-1}} \\
  d_1 & 0 & 0 & \frac{1}{2} & \ldots & \frac{1}{2^{d'-(d+\lfloor \log c \rfloor)-2}} \\
  \vdots & & & \ddots & \ddots \\
  d_{d'-(d+\lfloor \log c \rfloor)-1} & 0 & 0 & 0 & \ldots & 1 \\
  d_{d'-(d+\lfloor \log c \rfloor)} & 0 & 0 & 0 & \ldots & 1
\end{bmatrix}
\]
Let $p_{ij}$ denote the probability of transition from state $d_i$ to $d_j$. Here $d_i$ is the state with proofs of length $d + \lfloor \log c \rfloor + i$.

Therefore, $E[X_i] = 1 + \sum_{j=1}^{d'-(d+\lfloor \log c \rfloor)} p_{ij}E[X_j]$ (where $p_{ij} = \frac{1}{2^j-i}$), with $E[X_{d'-(d+\lfloor \log c \rfloor)}] = 0$. We are required to find $E[X_0] = 1 + \sum_{j=1}^{d'-(d+\lfloor \log c \rfloor)} p_{0j}E[X_j] = 1 + \sum_{j=1}^{d'-(d+\lfloor \log c \rfloor)-1} \frac{1}{2^j}E[X_j]$. On further unraveling the recursion (telescopic sum) we have $E[X_0] = 1 + (d-(d'+\lfloor c \rfloor))/2$.

Armed with the above lemma, we are now ready to bound the number of queries that the above oblivious attacker asks. Here we take $\varepsilon$ as the expected ratio of the number of elements in the database to $pDict$ (subject to the restriction that $D \subseteq pDict$), and not a concrete lower bound as in the definition.

**Theorem 3.2.5.** The oblivious attack can enumerate all the elements in the database, with confidence $\varepsilon = 1 - \frac{1}{2^{d'}}$ and success probability $\delta \approx 1$, running in time $O(2^d + m(d' - d))$.

**Proof.** We would first like to prove that every element is retrieved with a confidence of $\varepsilon = \frac{2^{d'}}{n}$ i.e., $E[|D|/|pDict|] \geq \frac{2^{d'}}{n}$.

We see that for every element in the database, the surrounding region is an interesting region. By construction of the Merkle tree, we know that every interesting region will have proofs of length $> d$. Therefore, our algorithm does not miss any potentially interesting region. Furthermore, we advocate a DFS of the pre-computed tree in the interesting region. Therefore, we should always enumerate all the “yes instances” in the region. Each ”yes instance” is of size at most $2^{k-d'}$. Therefore, the set of all “yes instances” cover at most $\frac{m}{2^{d'}}$ fraction of randomly generated queries. Therefore, $pDict$ will contain an expected $\frac{nm}{2^{d'}}$ elements, $m$ of which are definitely in the database. Therefore, we see that $|D|/|pDict| \geq \frac{2^{d'}}{n}$.

Now to bound running time, we see that we required $2^d$ many queries to enumerate all proof-buckets (with high probability, the negligible case when it doesn’t happen is if we directly hit an entry for the database or its sibling). Let us assume for convenience that there are at most $c$-many positive instances in every interesting region. Furthermore,
for retrieving every “yes instance” we see from lemma 3.2.4 that we require an expected $1 + (d' - (d + \lfloor \log c \rfloor))/2$ many queries. Therefore, for $m$ yes instances we require at most $m + m(d' - (d + \lfloor \log c \rfloor))/2$ queries (we might require fewer because, for some instances during the DFS we shall be starting with longer proofs). To account for the vestigial queries made during the DFS, note that for $c$ number of yes instances in an interesting region, there are $2c(d' - (d + \lfloor \log c \rfloor))$ many terminal nodes (for which we could ask a query) in the pre-computed Merkle tree. Therefore, there are at most $2m(d' - d)$ such vestigial queries made. Therefore, in total, we make an expected $O(2^d + m(d' - d))$ many on-line queries.

\textbf{Note:} We have so far not defined what a confidence interval signifies. Assume that on obtaining a dictionary, using the yes instances that have been computed thus far, an adversary would like to identify elements that fall in the database. We see that number of elements from the dictionary that are expected to fall in the bucket encapsulated by the yes instance is $\frac{n}{2^d}$. Thus, during the oblivious phase, the adversary can set $d'$ such that the expected number of elements in the yes instance is small. For example, if the adversary is fine with $\varepsilon = 1/2$, then $d'$ could be set to $\log n - 1$. 

\hfill \Box
Chapter 4

Oblivious Attacks and Epsilon Nets

In this chapter, we will present oblivious adversaries against Res belonging to certain function classes. We believe that these function classes are a generic representation of “efficient” responders. The two function classes discussed here are

1. The first is the set of all responders that use a set of proofs non-negligibly often. The non-existence of each domain can be proved by exactly one proof in the set. We believe these proofs are repeated non-negligibly often. This is the conventional notion of responders. Such responders perform minimal online computation, as the more frequent proofs can be precomputed. Both examples discussed in chapter 3 are responders from this function class.

2. The second is the set of all responders that contains a set of precomputed proof elements as its committed database CD. The proofs generated are subsets of CD. We do not restrict the size of the proofs, thus arbitrary subsets of CD could be valid proofs. Hence, there are $2^{|CD|}$ many possible proofs. This function class is a simplification of the function class where the responder doesn’t actively participate in data hiding. All privacy related computations are performed by the authenticator. Note that this simplification could be translated to proofs which use subsets of CD, but don’t perform computations to hide these subsets.

For both the above function classes we construct oblivious attacks, where the learner employs tools from computational geometry. In particular, the learner’s goal is to sample the “Epsilon Net” of the given function classes. We shall provide the preliminaries from
computation geometry in the subsequent section and then in later sections discuss the relation between an attack for the aforementioned function classes and epsilon nets.

4.1 Preliminaries

Epsilon nets were introduced as a concept in randomized construction of a data-structure to answer simplex range queries by Haussler and Welzl in [16]. All the definitions in this section are taken verbatim from [16] (with a few notational changes) unless mentioned otherwise.

**Definition 7.** A range space $S$ is a pair $(X, R)$, where $X$ is a set and $R$ is a set of subsets of $X$. Members of $X$ are called elements or points of $S$ and members of $R$ are called ranges of $S$. $S$ is finite when $X$ is finite.

Other works in the field also refer to range spaces as *hypergraphs*. We will refer to the same as range spaces and not hypergraphs.

Epsilon nets are defined over finite range spaces. They are simply samples of points that contain at least one member from all sufficiently large ranges.

**Definition 8.** Let $(X, R)$ be a range space, $A$ a finite subset of $X$ and $\varepsilon \geq 0$. Then $R_{A, \varepsilon}$ denotes the set of all $r \in R$ that contain a fraction of the points in $A$ of size greater than $\varepsilon$, i.e., such that $\frac{|A \cap r|}{|A|} > \varepsilon$. A subset $N$ of $A$ is an $\varepsilon$-net of $A$ (for $R$) if $N$ contains a point in each $r \in R_{A, \varepsilon}$.

We are interested in bounds on the size of such epsilon nets, i.e., $|Y|$. The bounds were proved in [16] by using a particular dimension on range spaces called the Vapnik-Chervonenskis dimension.

**Definition 9.** Let $S = (X, R)$ be a range space and let $A \subseteq X$ be a finite set of elements of $S$. Then $\xi_R(A)$ denotes the set of all subsets of $A$ that can be obtained by intersecting a range of $S$, i.e.,

$$\xi_R(A) = \{A \cap r : r \in R\}$$
If $|\xi_R(A)| = 2^{|A|}$, then we say that $A$ is shattered by $R$. The Vapnik-Chervonenkis dimension of $S$ (in short the VC-dimension of $S$) is the smallest integer $d$ such that no $A \subseteq X$ of cardinality $d + 1$ is shattered by $R$. If no such $d$ exists, then we see that the VC-dimension of $S$ is infinite.

An epsilon-net would be an useful tool if there were an efficient algorithm to construct one. Fortunately, a random sampling of points gives us an epsilon-net with high probability. The sample size is independent of the size of the domain. The theorem is again taken verbatim (with minor modifications for notational consistency and brevity) from [16, Corollary 3.8].

**Theorem 4.1.1.** For any range space $S = (X, R)$ of finite VC-dimension $d$, finite $A \subseteq X$ and $\varepsilon, \delta > 0$, if $N$ is the set of distinct elements of $A$ obtained by $k \geq \max\{\frac{4}{\varepsilon} \log \frac{2}{\delta}, \frac{8d}{\varepsilon} \log \frac{8d}{\varepsilon}\}$ random independent draws from $A$, then $N$ is an $\varepsilon$-net of $A$ for $R$ with probability at least $1 - \delta$.

### 4.2 Polynomial Proofs

Note that the size of $\mathsf{CD}$ is polynomial in $m$, where $m$ is the size of the database. Further, all efficient schemes will require that the size of $\mathsf{CD}$ is $O(m)$. This follows from the design consideration that DNSSEC shouldn’t add asymptotically significant overhead over DNS.

We will consider in this section, $\mathsf{Res}$ reuses certain proofs non-negligibly often. To elaborate, let $\Pi = \{\pi_1, \pi_2 \ldots, \pi_k\}$ be some set of proofs, where $k \approx \text{poly}(m)$. Then, $\forall \mathsf{CD}, \mathsf{PK}, \exists \Pi$, such that

$$\Pr_{\pi \in \Pi} [x \leftarrow \mathsf{DS} : \pi \leftarrow \mathsf{Res}(\mathsf{CD}, \mathsf{PK}, x)] > \varepsilon_k$$

where $\varepsilon_k$ is some system threshold (depending on the protocol), such that $\varepsilon_k$ is a non-negligible value in $m$. In short, $\Pi$ contains proofs that occurs with probability greater than $\varepsilon_k$. A graphical representation of such proofs is given Figure 4.1.

We make the following assumptions about our proofs here:
Figure 4.1: A graphical representation of repeated proofs. Every element in the domain space is colour/texture-coded with the corresponding proof elements. For example, the red-checkered proof proves the non-existence of an element $x$ in the red-checkered region in the domain-space.

1. Every proof element in $\Pi$ proves the non-existence of at least $\varepsilon$ fraction/region of the entire domain space. This region need not be a continuous space.

2. Given $D$ and $PK$, each domain name $x \not\in D$ maps to exactly one proof region i.e., falls in exactly one region. The proof of non-existence of $x$ is the proof $\pi$ associated to the said region. For example, if $x$ lies is the solid filled(orange-filled region) of the domain space in Figure 4.1 then the proof is the orange/solid-filled rectangle. In other words, the regions are disjoint.

3. We assume that $\Pr[x \xleftarrow{R} DS : \pi \leftarrow \text{Res}(CD, PK, x) | \pi \in \Pi] = \Pr_{\pi \in \Pi}[x \xleftarrow{R} DS : \pi \leftarrow \text{Res}(CD, PK, x)]$, i.e., adaptive choice of domains given their region is infeasible. In other words, the distribution of proofs that we obtain is independent of the distribution of the sample of domains.
4. We assume here that no $\pi \in \Pi$ also proves the existence of an element $x$ in the database. Even if it does, it proves the pair $(x, y)$ exists in the database, where $y = D(x)$. The reason for the same is if indeed, it were to prove that only $x \in D$ (and not $(x, D(x))$), then we could trivially run our dictionary on the proof $\pi$ to learn that $x$ is in the database.

NSEC and NSEC3 falls in such a class of functions. In a gist, both these classes create some ordering on $D$ along with a signature on adjacent pairs in the ordering. $\text{CD}$ contains these adjacent pairs as proof elements. The proof set $\Pi$ is the set of all elements in $\text{CD}$ that occur non-negligibly often, i.e., proofs where the range covered is sufficiently large. In the case of NSEC, $\varepsilon_k$ depends on the database at hand. For NSEC3, assuming random oracles, we can expect evenly distributed ranges, so we could set $\varepsilon_k \approx 1/\text{poly}(m)$ for some $\text{poly}(m)$.

For further details on NSEC3, we refer the readers to section 2.2 and for the corresponding enumeration attack we refer the readers to 3.2.1.

The hybrid scheme (refer 2.3 and 3.2.2), again falls in this category. In this case, the proof set $\Pi$ is the set of all paths of lengths at most $\log(1/\varepsilon_k)$ that represent proofs of non-existence of a range (based on the outputs of the random oracle) of domains.

4.2.1 Relation to Epsilon Nets

Our oblivious adversary as defined in chapter 3 consists of the learner and the pruner. The goal of the learner for this function class will be to learn to $\Pi$. The pruner in this case eliminates elements from $\text{Dict}$, if in $\Pi$ it can find the required proof of non-existence of $x$. In that case, by Assumption 3, then the expected number of elements pruner will be able prune from the dictionary, is at least $k\varepsilon_k$, i.e., pruner retrieves a pruned dictionary $\text{pDict}$, such that $|\text{pDict}| \leq m + k\varepsilon_k n$. Therefore, the oblivious adversary can prune to $\varepsilon = \frac{m}{m + k\varepsilon_k n}$.

Therefore, we need the learner to collect samples sufficient to retrieve $\Pi$. We show that the learner needs to sample an epsilon net in this case. Consider a range space defined as follows:
• Let $S = (DS \times CD, \{\pi : PK\})$ be the range space, where $CD$ is the space of all possible committed databases using the domains from the domain space $DS$. The set of all ranges is represented by $\{\pi : PK\}$. To elaborate, the range is nothing but the region of domains (given a database $D$) whose non-existence in $D$ is proved using the proof $\pi$.

• Every point in the range space is of the form $(x, CD)$, for some element in the domain and some committed database $CD$.

• The finite set $A$ for which we would like to determine an epsilon net is determined based on the input $CD$ of the Responder. Given, $CD$, $A$ is nothing but the set of all domains in $DS$. More formally, $A(D) = \{(x, CD) ; \forall x \in DS\}$. The set of all $\{\pi : \pi \mcap A\}$, is determined by $\exists x \in DS$ such that $\pi \rightarrow \text{Res}(CD, PK, x)$. Our assumption is that the ranges determined by $\pi$ is deterministic given $CD$. If random coins independent of $CD$ were to be used, it might contradict soundness. Note that, the verifier doesn’t need to explicitly specify $CD$ in this case, since it is implicitly included in $A$ through the responder’s input.

Note that, retrieving the proof $\pi$ just once, will give us the “knowledge” of all other elements in the range. We get the knowledge of all other elements, because, for a randomly sampled element $x$, we simply have to check that $\text{Ver}(x, PK, \pi) \rightarrow \text{Valid}$ (we mean some output that isn’t $\text{Invalid}$). By soundness of the proof system, with overwhelming probability, if valid, $x$ is in the range, else it isn’t.

An $\varepsilon_k$-net of $A(CD)$ will contain at least one sample for every proof $\pi$ that occurs with probability $> \varepsilon_k$, i.e., the proof set $\Pi$. Therefore, we need to sample an $\varepsilon_k$-net (according to definition 8) of domains and query on them.

Since, we assume that the proof regions/ranges (not the exact value of $\pi$) can be uniquely determined given $CD$, therefore, the VC dimension $d$ for $A(CD)$ is 1, i.e., $d = 1$. Now, from theorem 4.1.1, an $(\delta, \varepsilon, n, m)$-learner (where $\varepsilon$ is as set earlier) queries the responder with $O(\max\{\frac{4}{\varepsilon_k} \log \frac{2}{\delta}, \frac{8}{\varepsilon_k} \log \frac{8}{\varepsilon_k}\})$ randomly sampled domains, to retrieve $\Pi$.
with probability at least $1 - \delta'$, where $\delta' = 1 - \delta$.

Note that our assumption that $CD$ uniquely determines the ranges, isn’t very restrictive. In NSEC, the database $D$ specifies the ranges. Similarly, in NSEC3, we can consider $CD$ to be the hashed database, along with the initial (random) salt. The same holds with the pruned Merkle Tree of the hybrid scheme too. The assumption doesn’t disallow salts, but it does restrict to one salt per $CD$. If the responder is allowed to use additional randomness in an ad-hoc fashion to determine the region, soundness of the protocol may be adversely affected (like in NSEC3). For example, in NSEC3, if each proof is allowed a different salt, then for an overwhelming majority of salts, elements in the database $D$ will yet be covered by NSEC3 proofs.

### 4.3 Subset Proofs

In the previous section, we showed that no scheme which has proofs occur non-negligibly often is secure against oblivious attacks. In this section, we consider the possibility of exponentially many proofs. Note that to have exponentially many plausible proofs without any online computation, some pre-computed component has to used non-negligibly often to compute proofs. Therefore, we work with idea that each of these proofs have proof components that occur non-negligibly often.

To elaborate, we consider $CD$ to be a set of proof elements, and for every query $x$, the proof $\pi_x \subseteq CD$. Let $\Pi = \{c_1, c_2, \ldots c_k\}$ be the subset of proof components from $CD$ that are repeated non-negligibly often. Therefore, again, we have $\forall CD, PK, \exists \Pi$, such that

$$\Pr_{x \leftarrow DS, \pi \leftarrow \text{Res}(CD, PK, x), c \in \pi} : c \in \pi > \varepsilon_k$$

where $\varepsilon_k$ is some system threshold (depending on the protocol), such that $\varepsilon_k$ is a non-negligible value in $m$. In short, $\Pi$ contains proof components that occur with probability greater than $\varepsilon_k$.

A graphical representation of such proofs is given Figure 4.2.
Figure 4.2: A graphical representation of subset proofs. Every element in the domain space is texture-coded with the corresponding proof elements. A region can be covered by more than one component. The components covering a region are mapped to by arrows. The proof contains all components covering a region.

The primary considerations here are:

1. Every proof component in CD covers a region of the domain space (just as in the case of repeated proofs). CD consists entirely of proof components alone, there is no additional secret.

2. Every domain can be covered by arbitrary number of proof components. But the proof of non-existence of such a domain is the set of all components that cover the domain. For example, in Figure 4.2 for a point \( x \) in the checkered region, the proofs are nothing but the proof components represented by horizontal and vertical lines.

3. Given CD the region covered by each proof component can be uniquely determined, i.e., there is no additional randomness employed by the responder in determining the region.

4. This section again considers only negative proofs, i.e., the said set of proof elements
do not prove the existence of an element in the database.

5. \( \text{Res} (\mathcal{C}D', PK, x) \) will generate a valid proof if \( \mathcal{C}D' \) contains all proof components from \( \mathcal{C}D \) to generate the proof for \( x \). In other words, if \( \mathcal{C}D' \subset \mathcal{C}D \), \( \text{Res} (\mathcal{C}D', PK, x) \) will generate a valid proof for all \( x \) that falls in the ranges covered only by \( \mathcal{C}D' \).

The notion that the proofs are just sets of proof components is a simplification. This can be generalised to say that the proofs reveal this aforementioned set of components. This scenario will hold, when \( \text{Auth} \) is responsible for all data-hiding/privacy related computations. As claimed in [5], confidentiality isn’t a DNSSEC goal. Thus, \( \text{Res} \) would ideally perform minimal online computation for combining these components, and all confidentiality related computation would be delegated to \( \text{Auth} \) when it is generating those components.

We do not know of any protocols that fit this definition, but we feel it is the next logical step given that using proofs that cover large ranges of domains aren’t private.

### 4.3.1 Learner and epsilon-nets

Again, we would like a learner that enumerates \( \Pi \). The reasoning for the same will explained in the following section, where we discuss the pruner. Again, we attempt to visualize the proof system as a range-space. Again,

- Let \( S = (\mathcal{D} \times \mathcal{C}D, \{ c : PK \}) \) be the range space, where \( \mathcal{C}D \) is the space of all possible committed databases using the domains from the domain space \( \mathcal{D} \). The set of all ranges is \textit{represented} by \( \{ c : PK \} \). To elaborate, the range is nothing but the region of domains (given a database \( D \)) covered by proof components \( c \). The non-existence of \( x \) in \( D \) is proved using a set of these components.

- Every point in the range space is of the form \( (x, \mathcal{C}D) \), for some element in the domain and some committed database \( \mathcal{C}D \).

- The finite set \( A \) for which we would like to determine an epsilon net is determined based on the input \( \mathcal{C}D \) of the Responder. Given, \( \mathcal{C}D, A \) is nothing but the set of
all domains in \( DS \). More formally, \( A(D) = \{(x, CD); \forall x \in DS\} \). The set of all \( c \), such that \( c \cap A \) is \( c \in CD \) such that \( \exists x \in DS, c \in \text{Res}(CD, PK, x) \).

Given our assumption that \( CD \) uniquely determines the regions covered by every \( c \in CD \), we have VC dimension \( d = 1 \) for \( A(CD) \). If we eliminate this assumption, we don’t currently know a bound on the VC dimension.

Therefore, for a database \( D \), \( \Pi \), consists of all the oft-repeating proof components. Therefore, the learner would simply need to construct an \( \varepsilon_k \)-net, to enumerate \( \Pi \). For some parameter \( 0 < \delta \leq 1 \), the learner uses \( O(\max\{\frac{1}{\varepsilon_k} \log \frac{2}{\delta}, \frac{8}{\varepsilon_k} \log \frac{8}{\varepsilon_k}\}) \) randomly sampled domains, to enumerate \( \Pi \supseteq \Pi \) with probability at least \( 1 - \delta' \), where \( \delta' = 1 - \delta \).

4.3.2 Pruner and \( \Pi \)

The pruner in this case on receipt of \( \Pi \) must be able to appropriately verify if a domain \( x \) will lie in the ranges described by its proof components. Since, we assume that for all \( CD' \subseteq CD \), \( \text{Res}(CD', PK, x) \) can generate valid proofs for all regions covered exclusively by \( CD' \), we shall prune by simply running \( \text{Res}(\Pi, PK, x) \) for all \( x \in \text{Dict} \). If it generates a valid proof, we discard \( x \) from the dictionary.

For a reasonable \( CD \), there must exist an \( \Pi \) that prunes a non-negligible fraction of the elements. If not, then every proof requires at least one component that occurs with probability \( < \varepsilon_k \). This implies that \( CD \) contains at least \( 1/\varepsilon_k \) components. Thus, if \( \varepsilon_k \approx 1/n \), the Responder stores at least \( n \) components. Since \( n >> m \), the storage costs render the proof system inefficient.

Therefore, for a sufficiently low \( \varepsilon_k \), \( CD' \) should generate proofs for at least \( \varepsilon_k \) fraction of the entire region (i.e., the component set is overlaps on the same region). Thus, we have a \((\delta, \varepsilon, n, m)\)-oblivious attacker, where \( \varepsilon > \varepsilon_k \).

\(^1\)It will receive \( \Pi \supseteq \Pi \), but we assume for ease of analysis that \( \Pi = \Pi \).
Appendix

Zero-knowledge Databases

Zero-knowledge elementary databases were introduced in [24]. It is a two-party protocol, where the prover has the knowledge of an elementary database $D$ which consists of tuples of the form $(x, D(x))$. The verifier queries to the prover on some index $x'$ and the prover returns a verifiable response which is either $D(x')$ or the absence of $x'$ in $D$.

Formally, a ZK-EDB can be defined as follows [10].

**Definition 10.** A ZK-EDB for the security parameter $k$ on a simple database $D$ (consisting of tuples $(x, D(x))$), consists of the probabilistic polynomial time algorithms ZK-Setup, $P = (P_1, P_2)$, $V$ described as follows:

**ZK-Setup**($1^k$). It receives the security parameter as input, it outputs $\sigma$ (the CRS or the public parameters).

$P_1(1^k, \sigma, D) \rightarrow (Comm, T)$. $P_1$ on receiving the security parameter (as a unary string), the CRS and the database $D$ as input, halts after outputting a public commitment $Comm$ and an associated secret/trapdoor $T$.

$P_2(1^k, \sigma, Comm, T, x) \rightarrow \pi_x$. $P_2$ on receiving the security parameter, the CRS, the public commitment, the secret and a query $x$, produces a proof of $\pi_x$ on $D(x)$. (Note that, $D(x) = \bot$ if for $x$, there is no $d$ such that $(x, d) \in D$)

$V(1^k, \sigma, Comm, x, \pi_x) \rightarrow y$. $V$ on receiving the security parameter, the CRS, the public commitment, and an additional input of the query $x$ along with its proof $\pi_x$ outputs $y$. In this case $y = D(x)$ (if it believes $(x, D(x)) \in D$), $y = Out$ (if it believes that
for \( x \), there is no \( d \) such that, \((x, d) \in D\) or \( y = \text{Invalid} \) (if it finds that proof \( \pi_x \) is false).

\( P_1 \) is the committer to the database, \( P_2 \) is the prover and \( V \) is the verifier. The security properties of a ZK-EDB are listed as follows:

**Completeness** Formally, completeness is the requirement that, if \( \text{ZK-Setup}(1^k) \rightarrow \sigma, \ P_1(1^k, \sigma, D) \rightarrow (\text{Comm}, T) \) and \( P_2(1^k, \sigma, \text{Comm}, T, x) \rightarrow \pi_x \), then \( \Pr[V(1^k, \sigma, \text{Comm}, x, \pi_x) \rightarrow \text{Invalid}] \leq \epsilon(k) \) (where \( \epsilon(k) \) is some value negligible wrt \( k \)).

**Soundness** Formally, soundness is the requirement that, if \( \text{ZK-Setup}(1^k) \rightarrow \sigma \), then for all efficient algorithms \( P'(1^k, \sigma) \rightarrow (\text{Comm}', \pi_1, \pi_2) \) and \( \forall x \in \{0, 1\}^* \), \( \Pr[V(1^k, \sigma, \text{Comm}', x, \pi_1) \neq V(1^k, \sigma, \text{Comm}, x, \pi_2) \land V(1^k, \sigma, \text{Comm}', x, \pi_1) \neq \text{Invalid} \land V(1^k, \sigma, \text{Comm}, x, \pi_2) \neq \text{Invalid}] \leq \epsilon'(k) \) (where \( \epsilon'(k) \) is some negligible value wrt \( k \)).

**Zero-knowledge** Let the verifier (possibly malicious) \( V \) run in two phases, i.e., \( V = (V_1, V_2) \). Here \( V_1 \) takes as input the parameters generated and outputs the database \( D \), while \( V_2 \) is the interactive verifier (with additional input of some ”state” from \( V_1 \)). Then, there exists a simulator \( S \) with oracle access \( D \) with the restriction that \( S \) queries the oracle on \( x \) only when it is queried for that \( x \) by the verifier \( V_2 \) that simulates upto computational indistinguishability the view of the verifier \( V_2 \) when it interacts with the actual prover \( P = (P_1, P_2) \), for any database \( D \leftarrow V_1(\sigma, 1^k) \). The view (or state) of \( V_2 \) from the transcript \((\sigma, \text{Comm}, x_1, \pi_{x_1}, \ldots, x_n, \pi_{x_n}) \) obtained on interacting with \( P = (P_1(1^k, \sigma, D), P_2(\text{Comm}, T, \sigma, x)) \) is computationally indistinguishable from the view of \( V_2 \) from the transcript \((\sigma', \text{Comm}', x_1, \pi'_{x_1}, \ldots, x_n, \pi'_{x_n}) \) obtained on interacting with \( \text{SIM}^D(1^k) \), i.e., the distribution of the transcripts is identical upto computational indistinguishability.
Bibliography


[18] Dan Kaminsky. It’s the end of the cache as we know it. 2006.


