Secure searching through encrypted data

Creating an efficient Hidden Vector Encryption construction
using Inner Product Encryption

Dirk van Veen

July 2011

Management Summary

In this thesis, the problem of hosting a database with sensitive information on an honest but curious server is explored. Simply stated, the problem is the following: *how can we store information and perform queries in such a way that the server cannot learn the contents of the information or the queries.* Although in an ideal situation the stored information would be a relational database, the scope of this thesis is limited to the storage and querying of metadata.

Two of the general approaches to solving this problem are *Hidden Vector Encryption* (HVE) and *Inner Product Encryption* (IPE). The former allows for conjunctions of a wide variety of queries in a relatively efficient manner, but has some innate problems regarding the confidentiality of search queries; the latter allows for disjunctions of an even wider variety of queries and can prove confidentiality of search queries, but suffers some efficiency problems.

In this master thesis, a solution is constructed that incorporates both the efficiency of HVE and the security of IPE. In order to do this, current HVE and IPE implementations and the intuitions behind these implementations are studied to retrieve their innate strengths and weaknesses and to understand their interrelations. This creates a context, within which a main intuition is designed.

This intuition involves employing the main ideas behind Seghi's HVE construction to give IPE's shorter input vectors. This would make the computational cost of performing a query on a ciphertext linear to the maximum number of wildcards in a query instead of linear in twice the total number of elements in a query. Especially in HVE schemes that allow a relatively low number of wildcards, this would be a great improvement.

The intuition had one problem, however, as it was not fully compatible with existing IPE constructions.

Two attempts were made at creating a construction that compensates for this incompatibility.

The first attempt revolved around extending an existing IPE scheme, but was ultimately proven insecure. The second attempt uses one insight from the first attempt to amend the main intuition in such a way that the incompatibility is removed. This results in our main construction.

Additionally, the same technique that is used in the second attempt is used to create an alternative construction that is more efficient than the main construction in two specific situations.

Contents

1 Introduction **5**
1.1 General background 5
1.1.1 General introduction 5
1.1.2 Context 5
1.2 Searchable encryption 7
1.2.1 High level overview 8
1.2.2 Key settings 10
1.2.3 Predicate encryption 10
1.3 Problem description 11
1.4 Research Question 11
1.4.1 Security 12
1.4.2 Efficiency 12
1.4.3 Expressiveness 12
1.5 Approach 13
1.6 Layout and organization 14

2 Preliminaries **15**
2.1 Introduction to Algebra 15
2.1.1 Abstraction 15
2.1.2 Groups 16
2.1.3 Bilinear Maps 17
2.1.4 Group Order 18
2.1.5 Generators 19
2.2 Security 20
2.2.1 Formal proofs 21
2.2.2 Security proofs 21
2.2.3 Security types 23
2.3 Predicate types 23
2.3.1 Literals 24
2.3.2 Compositions 25

3 An analysis of predicate encryption **26**
3.1 Hidden Vector Encryption 26
3.1.1 Basics 26
3.1.2 Binary HVE 27
3.1.3 Constructions 29
3.2 Inner Product Encryption 32
3.2.1 Basics 32

3.2.2 HVE in IPE 33
3.2.3 Constructions 35

4 Results **37**
4.1 High level intuition 37
4.1.1 The approach 37
4.1.2 The intuition 38
4.1.3 Efficiency, expressiveness and security 40
4.2 First attempt 40
4.2.1 Intuition 40
4.2.2 Construction 41
4.2.3 Security 42
4.2.4 Result 45
4.3 Main construction 46
4.3.1 Intuition 46
4.3.2 Construction 47
4.3.3 Security 47
4.3.4 Efficiency 48
4.3.5 Expressiveness 48
4.3.6 Result 48
4.4 Alternative construction 49
4.4.1 Intuition 49
4.4.2 Construction 50
4.4.3 Security and Expressiveness 50
4.4.4 Efficiency 50
4.4.5 Result 51

5 Conclusion **52**
5.1 Future Research 53

List of Figures

1.1 Typical situation: database is hosted locally; server sees plaintext data. 6
1.2 Ideal situation: database is hosted externally; server only sees encrypted data. 7
1.3 Scope difference between categories 1 and 2. 8
1.4 Searchable Encryption algorithms within the ideal situation 9

Chapter 1

Introduction

This chapter contains the motivation and the setup of this master thesis. Starting with a general background and context description, it continues to explain the basics of searchable encryption and the problem within this domain that this thesis tackles. Having identified the problem, the chapter continues to formulate the main research question and the approach to finding an answer.

1.1 General background

1.1.1 General introduction

If you have paid attention to the media during the last decade, you have probably come across stories about “the need for cyber security”, or “the importance digital privacy”, or maybe even “the need for data protection strong encryption”. Clearly there exists a desire, some may say even a need, to protect information.

Luckily, there are almost as many solutions that provide ways to protect information as there are situations where this need is expressed. We can surf anonymously to a website, download files through a secured connection and store them in a hidden volume on an encrypted USB stick, which may only be accessed after two-factor authorization using a hardware token and our own biometrics.

There are many solutions, but there are still open questions.

1.1.2 Context

The specific context of this master thesis is the scenario where a database with sensitive information has to be stored on an *honest but curious* server, without losing the database’s query functionality.

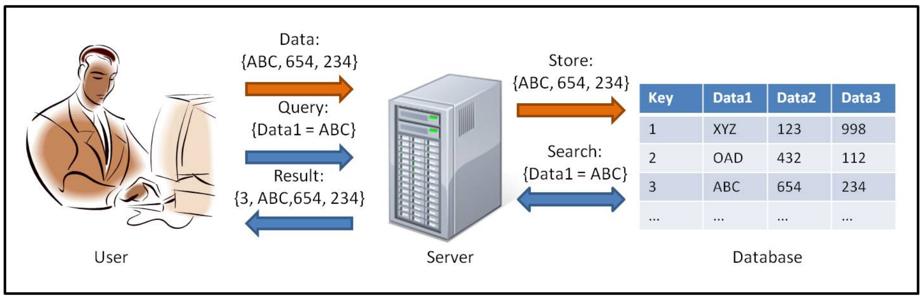
By “honest but curious” we mean that even though the server will stick to the protocol and it will not tamper with the integrity of our data (honesty), it may try to use any of the information it sees to learn and spread information about stored content or scheme-specific secrets (curiousity).

To illustrate this problem, imagine a company that uses a database to store its client administration. Because this company does not want to spend unnecessary resources on the database storage, they want to outsource the storage to a specialized hosting provider, that offers excellent availability and integrity guarantees (honesty) against very competitive prices. However, because the hosting provider offers the storage as a cloud service, with servers stationed around the world, it cannot guarantee that the data is stored in a country with the kind of privacy legislation that the company needs to meet its own privacy policy (so servers could be curious).

What the company would like in this scenario, is a way in which it can store and retrieve the information from the provider without exposing said information to the server. Ideally, the company would like some Solution, with which it could communicate like it would with a regular database, and which would handle information storage and retrieval in such a way that only the company would be exposed to the information.

Figure 1.1 shows typical interaction between a user and a locally hosted database. The orange arrows show how data is stored, the blue arrows how data is retrieved.

Figure 1.2 illustrates the same interaction within the ideal situation, with an externalized database that never sees any plaintext information.



Key	Data1	Data2	Data3
1	XYZ	123	998
2	OAD	432	112
3	ABC	654	234
...	...	...	...

Figure 1.1: Typical situation: database is hosted locally; server sees plaintext data.

Traditionally, the Solution consisted of allowing the server to decrypt portions of the database in memory and perform searches in memory while leaving all information on the hard disk encrypted. Within the context of a curious server, this approach becomes useless as the server cannot be trusted to keep the contents of its memory (or the decryption keys themselves) secret.

A simple solution to this problem would have the Solution store the database in an encrypted form at the hosting provider and retrieve, decrypt and re-encrypt the whole thing for every request by the company. Although this could work if the database is small, it does not scale and largely defeats the purpose of outsourcing the data storage in the first place.

Fortunately, a considerable amount of research has been performed recently concerning encryption schemes that try to tackle this problem. This has resulted in a wide variety of *searchable encryption* schemes, each with their respective advantages and disadvantages.

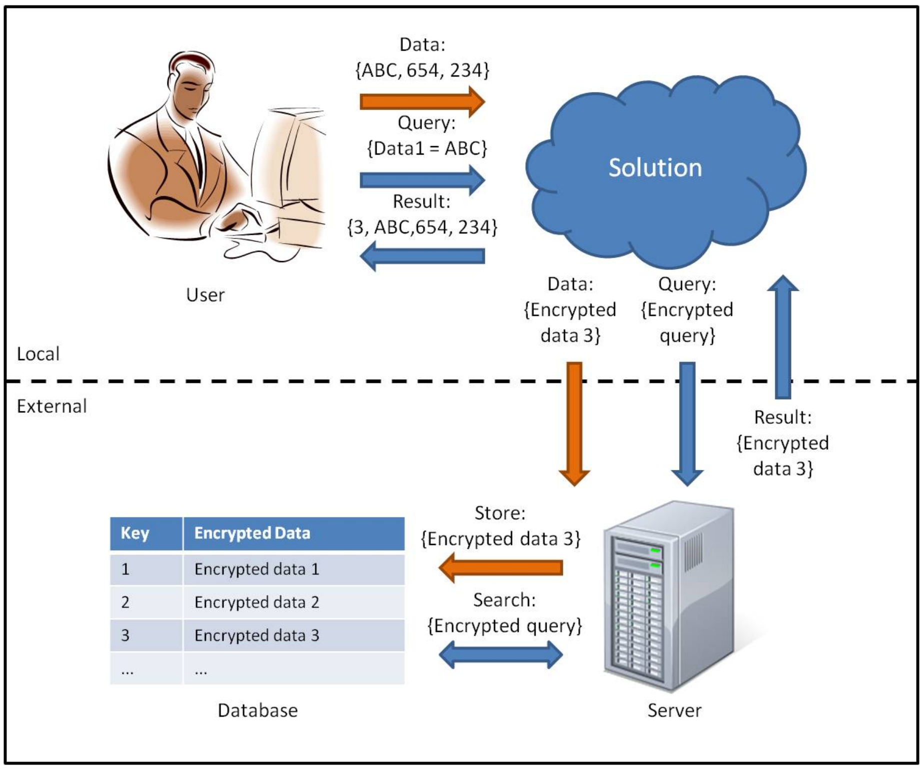


Figure 1.2: Ideal situation: database is hosted externally; server only sees encrypted data.

1.2 Searchable encryption

Searchable encryptions schemes can be roughly divided into two categories:

Category 1 Schemes that focus on searching through document content

Category 2 Schemes that focus on searching through document metadata

Category 1 schemes are designed to encrypt the full content of a document in such a way that every bit of information in it can be made queryable. The major advantage of this approach is that it allows for very detailed searches, making them perfect for content-oriented applications. However, this advantage comes at the cost of making the complexity of the search process dependent on the size of each individual document. This means that these schemes are not suitable for applications where documents may be of arbitrary sizes or where the number of documents to be searched should be allowed to continually grow.

Category 2 schemes are designed to only encrypt specific properties of a document and make these queryable. In these schemes, the actual content of a document is considered to be outside the scope of the scheme and is assumed to be safely encrypted before being stored. Instead, the user is allowed to define specific characteristic properties that can be used to identify documents. These properties are commonly referred to as a document's metadata, as they are information (data) about (meta) the document. Metadata may include the time of a document's creation, a document's author, a list of keywords related to the document's subject, a document's rubrication or any other type of user-defined information.

The major advantage of this approach is that it reduces and fixes the amount of information that needs to be queryable per document, so the complexity of the search process only depends on the number of documents being stored.

Figure 1.3 shows two documents with content and metadata. The blue rectangles denote the scope of category 1 schemes, the orange rectangles denote the scope of category 2 schemes. It can be readily seen that category 2 schemes scale much better than category 1 schemes, which is why this thesis focuses solely on category 2 schemes. From hereon, the term "searchable encryption" will be used synonymously with category 2 schemes.

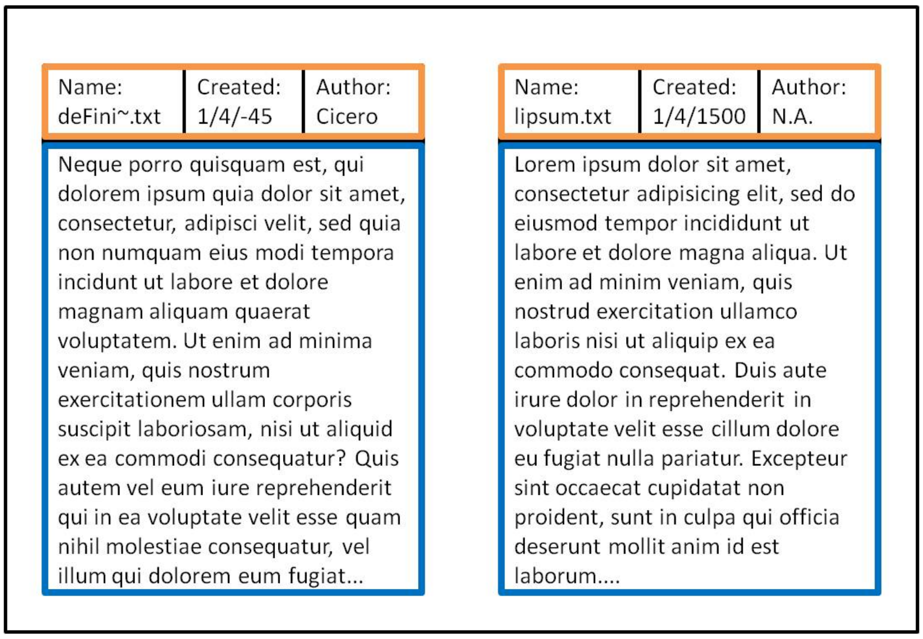

Name: deFini~.txt
Created: 1/4/-45
Author: Cicero
Neque porro quisquam est, qui dolorem ipsum quia dolor sit amet, consectetur, adipisci velit, sed quia non numquam eius modi tempora incidunt ut labore et dolore magnam aliquam quaerat voluptatem. Ut enim ad minima veniam, quis nostrum exercitationem ullam corporis suscipit laboriosam, nisi ut aliquid ex ea commodi consequatur? Quis autem vel eum iure reprehenderit qui in ea voluptate velit esse quam nihil molestiae consequatur, vel illum qui dolorem eum fugiat...
Name: lipsum.txt
Created: 1/4/1500
Author: N.A.
Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum....

Figure 1.3: Scope difference between categories 1 and 2.
(Blue represents the scope of category 1. Orange represents the scope of category 2.)

1.2.1 High level overview

Looking at searchable encryption schemes from a high level of abstraction, they usually consist of four parts, or algorithms.

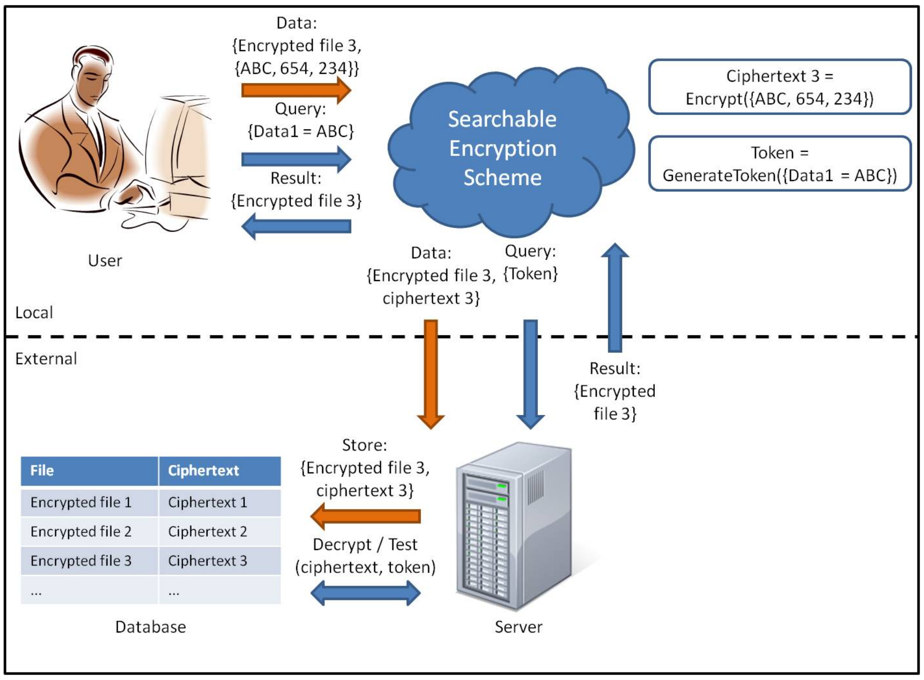
Setup This is the part that sets up the scheme and gives its participants the information they need top operate it.

Encrypt This is the part that encrypts metadata in such a way that it can be safely stored on an untrusted server.

Generate Token This is the part that transforms a search query into something unrecognizable called a token, which can be given to the server to identify which encrypted information matches the query.

Decrypt / Test This is the part that actually tests whether a piece of encrypted metadata matches the query that is represented by a token by trying to decrypt it with the token.

Figure 1.4 illustrates how these algorithms fit within the ideal solution.



File	Ciphertext
Encrypted file 1	Ciphertext 1
Encrypted file 2	Ciphertext 2
Encrypted file 3	Ciphertext 3
...	...

Figure 1.4: Searchable Encryption algorithms within the ideal situation

As mentioned before, a considerable amount of research has been performed on the creation of searchable encryption schemes, leading to a large diversity of schemes that fit the aforementioned construction. Broadly speaking, this diversity is the consequence of a trade-off between three factors:

Efficiency: This relates to the resource usage of the scheme, which can be expressed in a variety of ways. Common ways to measure the efficiency of a scheme include the sizes of ciphertexts and tokens, the number of computations needed for the various steps and the number of communication rounds between client and server during a search.

Expressiveness: This relates to the types of search queries which are supported by the scheme and the types of keys the scheme uses. Query types can be expressed by the types of predicates a scheme supports, the ways in which they can be combined and any possible limitations to the aforementioned inherent to the scheme.

Security: This relates to the types of security that a scheme can (or cannot) guarantee and the assumptions on which these guarantees rely. More information about the types of security will be given in 2.2.3.

1.2.2 Key settings

Just like regular encryption schemes, searchable encryption schemes come in (roughly) two types of key settings: the *Public Key Setting* (also known as the *asymmetric key* setting) and the *Private Key Setting* (also known as the *symmetric key* setting).

In the context of searchable encryption, a public key setting implies that:

- everybody who knows the public parameters can use the *Encrypt* algorithm to prepare and send documents for storage
- everybody who knows the public parameters can use the *Decrypt / Test* algorithm to evaluate prepared query tokens on such prepared documents
- only the holder of the secret key can use the *Generate Token* algorithm to create query tokens

A private key setting, on the other hand, implies that:

- everybody who knows the public parameters is able to use the *Decrypt / Test* algorithm
- only the holder of the secret key can use the *Encrypt* and *Generate Token* algorithms.

Although a public key setting can be useful for specific applications, such as the labelling of emails by the sender, it also has an inherent disadvantage: the content of a token can never be completely hidden. This has to do with the fact that a server is able to test the token on arbitrary ciphertexts [ABC$^+$08].

Private key schemes do not suffer from this inherent weakness and, although they may be highly impractical for applications that call for efficient key management, they are a perfect fit for the context of outsourcing databases, where the owner of the database should be the only party capable of adding or retrieving documents.

1.2.3 Predicate encryption

As current databases rely on expressive query languages like SQL, there is a demand for similar expressiveness in searchable encryption schemes. The subfield of searchable encryption that is focussed on providing such expressiveness is also known as *Predicate Encryption*, named after its aim to support any arbitrary type of predicate.

Two well-established Predicate Encryption primitives are *Hidden Vector Encryption* (frome hereon also referred to as HVE) and *Inner Product Encryption* (from hereon also referred to as IPE).

HVE schemes support all predicates that can be expressed as wildcard queries, which (through the use of clever data representation methods) includes conjunctions of (almost) arbitrary predicates. However, due to the way that all current HVE schemes rely on the plain text communication of the location of (non) wildcard elements (more on this later), they inherently suffer from information leakage in queries.

IPE schemes support an even wider range of predicates. Because IPE can be used to implement Hidden Vector Encryption (more on this later), IPE supports all of the predicates that are supported HVE. Additionally, IPE schemes support disjunctions, which makes IPE more flexible than HVE. IPE does not suffer from the aforementioned leakage problem and one of the IPE constructions has been proven to provide query privacy in a private key setting [SSW09]. However, all of this comes at a price concerning efficiency.

1.3 Problem description

The main problem that this thesis focuses on is the issue of efficiency.

As briefly mentioned in the subsection on predicate encryption, there is an efficiency gap between Hidden Vector Encryption and Inner Product Encryption. This gap is most evident when IPE is used to implement HVE. One of the most efficient regular HVE constructions [LL11] has a test algorithm that only needs a fixed three pairings. This contrasts heavily with the standard HVE implementation using IPE[1], from hereon referred to as *the KSW Approach* [KSW08], where inputs to the IPE algorithms are twice as large as their regular HVE counterparts, and the number of pairings in the test algorithm depends on those inputs linearly.

From an efficiency standpoint, HVE is a good candidate solution to the outsourced database problem. However, the fact that efficient HVE constructions inherently leak information about their queries[2] makes them unsuitable for this task.

IPE on the other hand, although less efficient, has at least one construction (from hereon referred to as *the SSW construction*[SSW09]) which has been proven to provide all the security features needed for such a task.

1.4 Research Question

The main research question that this thesis explores is the following:

Research Question How to construct a searchable encryption scheme with:

- the same security guarantees as the SSW Construction;
- greater efficiency than the KSW Approach in conjunction with the SSW Construction;
- at least the same expressiveness as HVE.

Although this formulation of the research question (in terms of the end result's comparison to other schemes) gives a clear indication of the research goal, it does not define how such comparisons can be made or measured. That is why in the next few sections, such definitions will be given.

Giving these definitions, however, introduces a slight dilemma, as it involves referring to properties that will not be explained until later in this thesis. This may make the

[1]This will be discussed in more detail in chapter 3

[2]Also discussed in more detail in chapter 3

definitions hard to understand. Still, it is better to include them here, both for completeness and for future reference. Therefore, do not be alarmed if some of the following terms may not seem familiar; they will be explained in due time.

1.4.1 Security

As mentioned in the research question, the solution should provide the same security guarantees as the symmetric SSW construction, which has been proven to be *selective single challenge secure*.

A solution is said to satisfy the security requirement if it offers at least both selective single challenge plaintext privacy and selective single challenge predicate privacy.

See section 2.2 for more information on security. See subsection 2.2.3 specifically for information on selective single challenge security.

1.4.2 Efficiency

The choice of how to measure efficiency is not a trivial one. Typical options include:

- size of ciphertext
- size of token
- number and type of computations during encryption
- number and type of computations during decryption
- mathematical context (e.g. group descriptions)

The use case we consider is that of databases being outsourced. As it is not unusual for current databases to be of a considerable size and the searchable encryption schemes need to test every entry in the database, the focus should be to minimize the cost of the test algorithm. Therefore, efficiency is taken as the computational cost of decryption. As pairings are considerably more expensive than regular group operations (multiplications), their number shall be used as the basis for comparison.

In the SSW Construction, the number of pairings necessary for decryption is $2N+2$, where N represents the length of the vectors, upon whose inner product the matching algorithm is based. For a standard IPE implementation of HVE, this N equals $2L$, where L represents the length of the HVE vectors.

Assuming a solution that supports HVE(-like) queries, it is said to meet the efficiency requirement if the *Decrypt / Test* algorithm uses fewer than $4L+2$ pairings, where L represents the length of a regular HVE vector.

1.4.3 Expressiveness

The expressiveness constraint says that a solution should be at least as versatile as HVE. Given that HVE is ultimately nothing more than simple wildcard matching, a solution is said to meet the expressiveness constraint if it can support wildcard queries.

1.5 Approach

The approach to solving the main research question consists of two steps:

Step 1: Analysis In this step, the concepts behind HVE, IPE and their respective constructions are studied for their respective strengths and weaknesses.

Step 2: Solving In this step, the results from the previous step are used to formulate and test intuitions for improved schemes or constructions.

This division into two steps is a natural result of the fact that the main research question, its goal and the constraints of the solution are defined through comparisons with HVE and IPE.

As both HVE and IPE are high level approaches that are not necessarily bound to specific mathematical constructions, the analysis in step 1 will contain both an analysis of the approaches in their high level form and of their current constructions.

The result of Step 1 can be found in chapter 3.

Because the development of a solution is a creative process and creative processes carry the risk of unnecessary time losses (e.g. through the pursuit of dead ends or through creative block) when left uncontrolled, the following cycle is used during Step 2:

1 Formulate an intuition based on the results from Step 1 (and if possible, of earlier iterations).

2 Guesstimate the feasibility of success within the time frame of a masters thesis. If success is highly unlikely within the set time frame, document the intuition and the reason that success will be unlikely, reject the intuition and start over.

3 Formulate a first concept construction of the intuition

4 Attempt to prove the concept construction secure under the constraints as defined in the problem description.

    I If successful, try to optimize the solution / minimize the strength of the assumptions used and document it

    II If unsuccessful, find out whether this is due to the intuition of due to the construction.

        a If the reason is that the intuition is flawed, try to prove it, document the failure and (if time permits) return to 1

        b If the reason is a construction error, try to fix the error and return to 4.

        c If running out of time, document the current results and extrapolate consequences for potential future research

The result of Step 2 can be found in chapter 4.

Following this approach should yield an account of the search process for an adequate solution, which either leads up to such a solution or at least offers an overview of failed approaches, including the reasons for failure, which may be used by future researchers to avoid pitfalls and dead ends. Additionally, if potential intuitions have been rejected based on the available time frame, a list of such intuitions can be used in future research as a source for initial inspiration.

1.6 Layout and organization

The rest of this thesis is organized as follows:

Chapter 2 Gives an introduction to the subjects of group algebra, security proofs and predicate types. This chapter is focused at giving an intuition on these subjects as they will be recurring throughout the thesis.

Chapter 3 Gives an analysis of both HVE and IPE, following Step 1 of the approach formulated in 1.5.

Chapter 4 Describes the results of Step of the approach formulated in 1.5. This includes a description of the main intuition, one failed iteration resulting in an insecure construction and one successful iteration resulting in a secure construction.

Chapter 5 Formulates a conclusion based on the based on the results.

Chapter 2

Preliminaries

This chapter provides readers with an introduction to some of the core concepts that appear throughout this thesis. These introductions are aimed at giving an intuition of these concepts, with a focus on readability.

Topics discussed include group algebra, bilinear mappings, security proofs and various predicate types.

To realize an intuition for all of these fields within the context of a master's thesis, a lot of formal definitions and technical details are skipped. This means that, although the explanations will give an intuition of what is going in the rest of the thesis, they by no means offer a qualified introduction into the respective topics. For qualified introductions, the reader is referred to their local library.

2.1 Introduction to Algebra

2.1.1 Abstraction

Mathematical expressions often take the following form :

$$A \odot B = C$$

Where A, B, and C may be either values or other expressions.

In such an expression, the $\odot$ represents some operation and is called the operator, while A and B represent the the elements that are subject to the operations and are called the operands. Common operations and their respective operators are additions $(+)$, subtraction $(-)$, and multiplication $(*)$.

Algebra is the field of mathematics that uses abstract properties and relations of operations to deduce new rules or schemes that apply to all operations with similar properties.

For example, when considering the simple example of adding numbers, algebra says that the operation *addition*, denoted by the operator $+$, has some properties known as *associativity* and *commutativity*:

$$\begin{aligned} \text{associativity:} \quad & (A+B)+C = A+(B+C) \\ \text{commutativity:} \quad & A+B = B+A \end{aligned}$$

It is easy to see that there are other operations that share these properties (such as multiplication), but that there are also operators that do not (such as division).

By focusing on the abstract properties, algebra can formulate rules and formulae that apply to all operations that share those specific properties. For instance, by combining the properties of associativity and commutativity, we can deduce that for every operation that is both associative and commutative the following holds:

$$A \odot B \odot C = C \odot B \odot A$$

This is a direct consequence of the aforementioned properties

$$\begin{aligned}
A \odot B \odot C &= (A \odot B) \odot C && \text{(associativity)} \\
&= (B \odot A) \odot C && \text{(commutativity)} \\
&= B \odot (A \odot C) && \text{(associativity)} \\
&= B \odot (C \odot A) && \text{(commutativity)} \\
&= (B \odot C) \odot A && \text{(associativity)} \\
&= (C \odot B) \odot A && \text{(commutativity)} \\
&= C \odot B \odot A && \text{the new rule}
\end{aligned}$$

Of course, this 'rule' can be 'discovered' for individual operations through trial and error, but using algebra that is no longer necessary. It provably applies as soon as the operation is both associative and commutative. That is the power of abstraction.

2.1.2 Groups

Just like algebra uses abstract properties to describe operations, it uses abstract properties to describe the operands and their relation to the operation. To do this, algebra uses a few different constructions, the most basic of which is the *Group*.

Basically, a Group is a combination of two things:

- a set of elements
- an operation that can take elements from the set as operands.

Of course, not just any set and any operation form a Group. There are a few prerequisites that have to hold before the elements and the operation may be called a Group:

- Every combination of two elements in the set using the operator should result in another element of the set
- The operation should be associative (as explained above)
- There should be an element ε in the set which, when combined with any other element a should produce that same element a. This element is also known as the identity element.
- For every element in the set, there should be an element in the set that can act as its inverse, i.e. with which it can be combined to produce the identity element.

As a practical example, consider the set of integers $\{\dots,-3,-2,-1,-0,1,2,3,\dots\}$ together with addition

- Any addition of two integers is itself another integer;

- Associativity is known to hold for addition of integers;
- For any integer a, $a + 0 = a$, so 0 is the identity element;
- For any integer a, $-a$ is also an integer and $a + (-a) = 0$, so every element has an inverse.

In a similar fashion, it can be shown that the set of all integers multiplied by 2 (that is $\{\dots, -6, -4, -2, 0, 2, 4, 6, \dots\}$) together with addition is also a Group (give it a try).

Although the group of integers under addition has an infinite size, there is no reason that groups cannot have a finite size. A simple example of groups of finite size is, for any integer $x \geq 1$, the integers modulo x under addition.

It is important to realize that, although the groups mentioned above are all sets of integers under addition, the definition of a group does not specify what an operation should be, or even that the members of the set in question should be numbers[1]. However, because the set elements often are numbers, the group notation can become a bit confusing. The reason for this is that, although a group only knows a single operation, application of the operation on multiple instances of the same element is usually shortened in some way.

In the two most wide spread notation styles, the additive (Abelian) notation and the multiplicative notation, this is done in the following way:

- Additive
  - application on distinct elements a and b: $a + b$
  - repetitive application on single element a: $a + a + a = 3a$
- Multiplicative
  - application on distinct elements a and b: $a \cdot b$
  - repetitive application on single element a: $a \cdot a \cdot a = a^3$

The important thing to remember here is that, although a and b are set members and therefore can be combined with other set members, the 3 in the above examples is just a shorthand for repetition and cannot be treated as another set element (even when the group set may contain an element 3).

Taking into account the above, it may be evident that mixed usage of the additive and multiplicative styles can lead to confusing situations. That is why in this thesis, only the multiplicative style will be used.

2.1.3 Bilinear Maps

Bilinear maps are, like the name implies, mapping functions. That means that they take elements from one or more different groups and associate them with one or more elements from another group. To be more specific, bilinear mapping functions associate two input elements with a single output element from a different group. In some cases the input elements come from a single group, in which case the mapping is called symmetric. In other cases, the input elements come from different groups, in which case the mapping is called asymmetric.

What makes bilinear mapping functions special is the term *bilinear*, which means that the the function is linear in both of its input elements. So what does that mean?

[1]The interested reader could look up dihedral groups, whose set elements are polygons

Practically, it means that if the input elements grow linearly (in an additive notation; exponentially in a multiplicative notation), then so does the output element. To put it in a formula (using the multiplicative notation), with e representing the mapping function

$$e(a^x, b^y) = e(a,b)^{xy}$$

This behavior is useful because it allows some problems that are supposed to be hard in one of the input groups to be solved in the output group.

For example, it is generally a hard problem to tell whether some element t is completely random or of the form a^{xy}, when you only know a, a^x and a^y. However, with a bilinear map, this check is quite easy:

$$e(a^x, a^y) = e(a,a)^{xy}$$
$$e(a^{xy}, a) = e(a,a)^{xy}$$

2.1.4 Group Order

Each group has an *order*, which is the number of elements in its set. The reason that it is mentioned here, is that for any group of order n and any element x inside that group's set, $x^n = e$, where e is the identity element (as explained above). For example, in the group $(\{0,1,2,3,4\},+)$ (that is, the integers modulo five, under addition), we have that the order of the group $n = 5$, $e = 0$, and for any of the elements x, $x^n = (x+x+x+x+x) \bmod 5 = 5x \bmod 5 = 0$.

In the previous example, the order of the group was a prime number. If the order n is not prime then it can be written as a product of prime numbers $n = p_1 \cdot \ldots \cdot p_i$, and the group is said to have a *composite* order. In such a group, it is possible to define subgroups whose order is a combination of one or more of the order's prime factors.

The reason for this can be explained as follows. Suppose that there is a group $(\{x_1, \ldots, x_{n-1}, \varepsilon\}, \odot)$ of an order n that can be rewritten as $n = ab$. Then for every element x in that group, it holds that $x^{ab} = \varepsilon$. However, because x^{ab} can be rewritten as $(x^a)^b$, you can say that there is a subgroup of order b that looks like $(\{(x^a)^1, (x^a)^2, \ldots, (x^a)^{b-1}, \varepsilon\}, \odot)$.

After all, the operation is the same as in the larger group so all of its prerequisites hold; the set contains the identity element; and every combination of elements $(x^a)^c \odot (x^a)^d$ is contained within the set.

To see this last property, note that the combined element is determined by the exponent $i = c+d$, which always can be rewritten as $i = r+qb$ with $r \leq b-1$, so the combination can be rewritten as $(x^a)^r \odot (x^a)^{qb} = (x^a)^r \odot ((x^a)^b)^q = (x^a)^r \odot (e)^q = (x^a)^r$ with $r \leq b-1$.

Supposing that either a or b can be further factorized, such that $n = abc$, the same logic can be used to define subgroups of order a, b, c, ab, ac and bc. This continues until n is completely factorized into primes.

Within the setting of bilinear mappings, there is another curious property that is related to group order. If the group in question has a composite order $n = p_1 \cdot \ldots \cdot p_i$ (all p prime) and we define subgroups G_a and G_b of orders a and b respectively as before, then if a and b do not share a common factor p_x, pairing two elements from these different subgroups results in the identity element of the target group ε_T. So, for any $g_a \in G_a, g_b \in G_b$ we would have that $e(g_a, g_b) = \varepsilon_T$

The reason for this stems from two things. First, the fact that with bilinear mappings, the order of the target group is the same as the order of the input groups, i.e. n.

Second, the fact that if the two subgroups do not share a common prime factor, then their elements can be rewritten such that the product of their exponents will contain all prime factors of n. [2]

To illustrate, suppose $n = p_1 p_2 p_3$, $a = p_1$ and $b = p_2$, then for any $g_a \in G_a$ and $g_b \in G_b$:

$$\begin{aligned}
e(g_a, g_b) &= e((x^{p_2 p_3})^c, (x^{p_1 p_3})^d) \\
&= e(x^{p_2 p_3 c}, x^{p_1 p_3 d}) \\
&= e(x,x)^{p_1 p_2 p_3 p_3 cd} \\
&= (e(x,x)^{p_1 p_2 p_3})^{p_3 cd} \\
&= (e(x,x)^n)^{p_3 cd} \\
&= (\varepsilon_T)^{p_3 cd} \\
&= \varepsilon_T
\end{aligned}$$

As a corollary, using different subgroups it is relatively easy to add a sort of “noise” to elements that can be removed with a pairing:

Suppose we have a group of order $n = abc$ with subgroups G_a, G_b and G_c, two secret values s_1 and $s_2 \in G_a$ and some public test value $T = e(s_1, s_2)$. Now suppose we want to communicate s_1 and s_2 without exposing their values such that the test value can still be evaluated. In that case, we can take random elements $B \in G_b$ and $C \in G_c$ and communicate the values $x = s_1 B$ and $y = s_2 C$.

As long as B and C are indeed random, the values of x and y should be sufficiently random to hide the values of s_1 and s_2. At the same time, their pairing still gives the same result:

$$\begin{aligned}
e(x,y) &= e(s_1 B, s_2 C) \\
&= e(s_1 B, s_2) \cdot e(s_1 B, C) \\
&= e(s_1, s_2) \cdot e(B, s_2) \cdot e(s_1 B, C) \\
&= e(s_1, s_2) \cdot e(B, s_2) \cdot e(s_1, C) \cdot e(B, C) \\
&= e(s_1, s_2) \cdot \varepsilon_T \cdot \varepsilon_T \cdot \varepsilon_T \\
&= e(s_1, s_2)
\end{aligned}$$

This masking using different subgroups is extensively used in predicate encryption schemes.

2.1.5 Generators

The last thing to mention, although some would argue that it should be the first, is the concept of *generators*.

When the complete set of a (sub)group of order n can be expressed as the powers of some element g (i.e. $\{g^1, \ldots, g^{n-1}, \varepsilon\}$, as was done in the creation of subgroups), then g is called a *generator* of that (sub)group.

[2] If the orders a and b share a prime factor, then rewriting the elements from the respective subgroups will show that the shared prime factor is missing.

If $a = \prod_{i \in J_a} p_i$, then all $g_a \in G_a$ can be rewritten as $(x^{m_a})^c$, with $m_a = \prod_{i \notin J_a} p_i$.

If b is described in a similar way and a and b share a prime factor x_s, then $s \in J_a$ and $s \in J_b$, so neither m_a nor m_b will contain prime factor x_s, which means their product is not divisible by n.

This is assuming n consists of distinct prime factors. Otherwise, a and b could be chosen such that their product does contain a prime factor x_s. This, however, would not change the divisibility by n as divisibility by n would still require an additional multiplication with x_s.

Generators are useful elements because of two things:

- For any set that can be defined as $\{g^1, \ldots, g^{n-1}, \varepsilon\}$, the operation $\odot$ acts in the same way, regardless of the value of g: $g^a \odot g^b = g^{a+b}$.
- It is possible for a single group to have multiple generators.

First of all, this means that generators can be used to define algebraic schemes which will hold for any group with a generator, simply by defining all elements in the scheme by their relation with the group's generator.

Secondly, this means that even though two implementations of the same function may perform exactly the same calculations on an algebraic level, the values of the actual elements used may be completely different when different generators are used. This is important, as it means that elements of different constructions can only be meaningfully exchanged if they agree on the generator(s) being used. This problem will return in 4.2.

2.2 Security

Whether a cryptographic scheme can be considered secure or not is determined by several factors, two major ones of which are:

- Correctness of the scheme
- Hardness of breaking the scheme

The *correctness* property says that a scheme does not produce false positives or false negatives. In the context of searchable encryption, this means that the scheme is constructed in such a way that the Decrypt / Test algorithm always manages to correctly determine whether a ciphertext's plaintext matches a token's query.

If a scheme is not correct, then it may happen that a token is used to retrieve more information than it should, possibly leading to information leakage.

Proof of correctness consists of two things:

- Showing how a correct decryption is derived in the case of matching ciphertext and token
- Enumerating the cases in which a false positive or negative might occur and showing that the probability of such a case is below some predetermined acceptable level.

Showing the *hardness of breaking the scheme* is a bit more complex. This is because there are various interpretations of both what is considered hard and what is considered breaking a scheme.

To deal with this problem, schemes often explicitly declare their interpretations in the form of hardness *assumptions* and security *theorems*, which are consequently proven in similar fashion to regular formal proofs.

2.2.1 Formal proofs

The essence of formal proofs is simple: starting out with some *assumptions* (axioms), and a *theorem* that you want to prove, you either try to derive the theorem directly from the assumptions or you try to show that if the theorem is false, then one or more of the assumptions must be false too (contradiction).

For example, say we have the following assumptions:

- everything that uses math is too hard to understand
- algebra is a part of math
- Inner Product Encryption uses algebra

If you now want to prove the theorem that you cannot understand Inner Product Encryption, then you could argue one of the following:

Direct derivation Because algebra is part of math and all math is too hard to understand, algebra is too hard to understand; because Inner Product Encryption uses algebra and algebra is too hard to understand, Inner Product Encryption is too hard to understand

Proof by contradiction If Inner Product Encryption weren't too hard to understand, then because Inner Product Encryption uses algebra, algebra wouldn't be too hard to understand; if algebra wouldn't be too hard to understand, and all math would be too hard to understand, algebra couldn't be part of math; but algebra is assumed to be a part of math, so there is a contradiction if we assume Inner Product Encryption is not too hard to understand

2.2.2 Security proofs

In the world of security proofs, both assumptions and theorems tend to have a specific format.

Security assumptions

Security assumptions can take one of two forms:

Computational We assume that given information I, value x is very difficult to compute

Decisional We assume that given information I about a situation and the fact that the situation could belong to one of the following two scenarios, it is very difficult to correctly guess the right scenario

Where the definition of "very difficult" usually means that there is no significantly better way of finding an answer than to just start guessing at random. Usually, such assumptions (also known as hard problems) are borrowed from mathematical fields like algebra.

Note that there is a relation between computational and decisional assumptions. If it is easy to compute some secret value x, then any decisional problem based on the value of x is trivial. This means that if a decisional problem based on x is provably hard, then the computational problem of finding x must be hard too (and may be even harder!). This is why most security proofs focus on decisional assumptions.

Security theorems

Security theorems are usually defined in the context of so called *games*.

Just like regular games, a security game (challenger) sets a goal for the player (attacker) to achieve and a few rules that have to be obeyed in trying to reach the goal.

Just like with the assumptions, goals are often of a decisional nature, asking the attacker to make a guess about which of two scenarios (usually decided by a fair coin toss) the game was played in. However, where assumptions are usually kept as generic as possible, the goal of a security game is usually about one or more specific components of the scheme. For instance, a common goal is to guess the correct metadata vector given two vectors and a ciphertext.

The rules of the game define the implicit restrictions placed on an attacker. Often these pertain to which algorithms may be accessed when and in what manner by the attacker. For instance, the rules may state that an attacker can use the GenerateToken algorithm to generate tokens for any query that doesn't match some ciphertext, or that the Encrypt algorithm may be used only once and only before the first token has been generated.

Typically, the rules of a security game shape a game such that it fits the following format:

Initialization The initial phase, in which the challenger and adversary define the context of the game

Setup The phase in which the challenger sets up the encryption scheme

Query I The first phase in which the attacker can query algorithms as specified by the rules

Challenge The phase in which the challenger presents the attacker with one or more instances of the decisional problem (often based on some input from the attacker)

Query II The second phase in which the attacker can query algorithms as specified by the rules

Response The phase in which the attacker gives its guess concerning the decisional problem

Generally a theorem states that, given one of the assumptions, the chance that any attacker outputs the correct guess and wins the game is not significantly greater than that given by the fair coin toss.

Proofs

Actual proofs of a theorem are then given using a proof by contradiction. In order to provoke such a contradiction, the game is set up in such a way that it does two things:

1 It requests a challenge from the given assumption's hard problem and sets up the encryption scheme in the same mathematical context (same group order, same generators, etc.)

2 It uses the assumption's challenge to effectuate the difference between the two scenarios of the game's own decisional challenge.

This way, if an attacker manages to discriminate between the two scenarios, he implicitly distinguishes between the two options of the assumed hard problem, which forms a contradiction with the assumption.

2.2.3 Security types

Security types are promises that can be defined by two things:

- What is actually promised (secrecy, indistinguishability, etc.) about which component (plaintext, ciphertext, token, etc.)
- The conditions under which the promise holds

Taking these in the context of games, the actual promise is reflected in the goal of the game, while the conditions are reflected in the rules.

Within the predicate encryption, there are two main promises of interest:

Plaintext privacy The promise that a ciphertext does not leak any information about the metadata vector it was made for.

Predicate privacy The promise that a token does not leak any information about the query vector it was made for.

Although the ultimate goal of any cryptographic scheme is to provide plaintext and/or predicate privacy without having to place any restrictions on the attacker, such a *full security* is often difficult to prove. This is why many constructions put specific restrictions on the abilities of an attacker, such that the scheme can be proven to provide plaintext and / or predicate privacy and that the restrictions can be argued to be acceptable.

Two of such restrictions that return in this thesis revolve around the number of times that the attacker may request a challenge and the time that the attacker has to provide its input for the challenge.

Single challenge security The attacker is limited to requesting a single instance of the decisional problem.

Selective security The attacker has to choose its input(s) for the challenge phase beforehand and provide them to the challenger during the Init phase (so that they may already be used during the Setup and Query I phase).

Single challenge security is used in the SSW IPE construction, as it can be shown that a single challenge secure IPE scheme for vectors of size $2N$ can be used to provide a fully secure IPE scheme for vectors of size N.[SSW09]

Selective security originally comes from the field of Identity Based Encryption (IBE) is used in a variety of HVE and IPE constructions. The motivation behind selective security is that any selective secure IBE scheme can be transformed into a non-selective (fully) secure IBE scheme (even though this transformation is not efficient).[BB04] The reason that this notion of selective security is carried over from IBE to various predicate encryption schemes is that predicate encryption schemes can be considered to be elaborate (anonymous) IBE schemes, with metadata vectors functioning as identities.

2.3 Predicate types

Predicate types are grouped into two categories: *literals* and *compositions*.

2.3.1 Literals

Literals are the basic building blocks of a predicate encryption scheme and correspond to the evaluation of *Boolean* comparison functions over a single metadata variable and a query variable. [3]

Three distinctive types of literals are distinguished

- Equality comparison
- Ordered comparison
- Set comparison

Equality comparison

The equality predicate is the simplest predicate and checks whether a metadata variable is equal to the query variable. A special case of the equality predicate is wildcard equality, where either the variable or the value may contain special "don't care" elements called wildcards, which will match any other single element. For example, if $\star$ denotes the wildcard element and the predicate is used for string comparison, where a string is represented as a vector of character elements, the search string "a$\star$c" (represented as $(a, \star, c)$) should match both "abc" and "aqc", but not "abbc". Similarly, if the predicate is used to compare tuples of keywords, where each keyword is represented by a single element, the tuple (email, secret, $\star$) should match both (email, secret, wife) and (email, secret, mistress).

Ordered comparison

Ordered comparison predicates compares whether the metadata variable is or is not greater than the query variable based on some scheme supported ordering mechanism (i.e. $metadata > value$ for some definition of $>$).

A special case of the comparison predicate occurs when either the metadata variable or the query variable represents a range, in which case the ordered comparison checks whether the range includes the compared variable (i.e. whether $rangeEnd > comparedVariable > rangeStart$ for some definition of $>$). Note that this means that range queries can be implemented either with the range in the token or with the range in the ciphertext. This choice reflects the choice of access policy location in ABE: range in ciphertext corresponds to CP-ABE and range in token corresponds to KP-ABE[4]

Set comparison

Set comparison predicates are comparisons where at least one of the two input variables is a set (the encompassing set) and determines the set membership of the other (the compared variable). When both variables represent sets, set membership may be

[3]Note the emphasis on *Boolean*, indicating that this overview only discusses functions who return either *True* or *False*. This is because, within searchable encryption, most queries can be reduced to a logical combination of such functions. Readers who are interested in queries that cannot be reduced in such a way are referred to the more general field of *Functional Encryption*, of which Predicate Encryption is a special case. There are currently no concrete schemes that allow for non-Boolean Functional Encryption, but there have been some advances regarding the privacy notions and (im)possibilities that are native to this subject [BSW10, O'N10].

[4]Note that this correspondence works both ways, and that any access policy that can be expressed in an anonymous ABE, could be translated to a predicate in a predicate encryption scheme.

interpreted either as the compared variable being a subset of the encompassing set or as having a non-empty intersection. Similar to the range query, the subset predicate can be implemented with the encompassing set in the ciphertext or with the encompassing set in the token. Subset in the token allows for versatile matching, whereas subset in the ciphertext allows for broadcasting mechanisms, as alluded to in [BW07].

2.3.2 Compositions

Compositions refer to logical combinations of other predicates, which may be either literals or compositions themselves. Compositions can be divided into two categories.

- Conjunctions
- Disjunctions

Conjunction

Conjunction predicates evaluate a Boolean AND combination of predicates. This means that a conjunction holds iff every one of its sub predicates holds. A special case of conjunction predicates are predicates in the Conjunctive Normal Form (CNF). CNF means that the predicate is a conjunction of one or more compositions, where each composition is a disjunction of literals (i.e. they do not contain conjunctions or other compositions).

Note that there is a difference between evaluating a conjunction of predicates and simply evaluating them individually (and combining the results), as conjunctions are evaluated as a whole and do not release information about individual clause(s) or literal(s) upon failure.

When an encryption scheme supports conjunctions of a variable number of literals (i.e. the number of literals in a conjunction is not fixed a priori by the number of distinct metadata variables), then conjunction queries can be used to create a trivial implementation of the wildcard query. This is done by creating a conjunction query over the non-wildcard elements: Given an encryption scheme that uses a tuple of three elements to represent metadata $(metadata_a, metadata_b, metadata_c)$, a wildcard query for $(element_a, \star, element_c)$ could be implemented as the conjunction $(metadata_a = element_a) \wedge (metadata_c = element_c)$.

Disjunction

Disjunction predicates evaluate a Boolean OR combination of predicates. This means that a disjunction holds as soon as one of its sub predicates holds. A special case of disjunction is the Disjunctive Normal Form (DNF). Similar to the CNF, DNF means that the predicate is a disjunction of one or more compositions, where each composition is a conjunction of literals. Again, note that there is a difference between evaluating a conjunction and evaluating individual predicates, as evaluating a conjunction does not release information about individual clause upon success.

Chapter 3

An analysis of predicate encryption

This chapter provides the results of the first step of the approach defined in section 1.5, i.e. an analysis of HVE and IPE. The former is a form of encryption that supports all queries that can be expressed as a wildcard query, which includes conjunctive queries over any preset collection of arbitrary predicates. The latter is a form of encryption that supports all queries that can be expressed as the evaluation of the inner product of two vectors. Together, they form the main approaches towards predicate encryption.

In the following two sections, HVE and IPE and their relation are discussed in more detail. This is done by analyzing each approach on the following three subjects:

1 Its high level approach and intuitions, which are construction-independent;

2 How it can be applied to achieve high expressiveness;

3 The characteristics of actual constructions.

3.1 Hidden Vector Encryption

3.1.1 Basics

Hidden Vector Encryption (from here on denoted as HVE) was first proposed in [BW07] and is, in essence, a simple wildcard encryption scheme. What this means is that search queries are not restricted to exact matching, but that they may include special wildcard characters $\star$ in the query that will match any one element in the metadata. That is, metadata and queries are somehow represented as a collection of elements and metadata is said to match a query if the metadata and non-wildcard query elements correspond.

For instance, by representing the keyword "bat" as a collection of three letters, a wildcard query could use the wildcard element $\star$ to search for similar words by ignoring specific characters . That is, to find all keywords starting with "ba" the query would "ba$\star$", or to find all keywords with an "a" as the middle character the query would be "$\star$a$\star$". Similarly, by interpreting a document description "type: email, sensitivity: top-secret" as a collection of two elements representing type and sensitivity, a wildcard query "email, $\star$" would match emails of every sensitivity level.

To put it a bit more formally: in HVE *metadata attributes* are encoded in an *attribute vector* (the collection) and predicates are encoded in a *query vector* of equal alignment (and consequently of equal length), where each vector contains *elements* from a scheme-specific *alphabet* Σ. In the first example, this Σ simply represents the 26 letters in the alphabet and can be chosen as the range $(1\ldots26)$. In the second example Σ is a bit more complex, as it represents both the range of possible document types and the range of sensitivity levels. In this case, Σ could be either a large single-purpose range of numbers, where each number represents a type or a sensitivity (i.e. $(1\ldots x)$ for all x types and $(x+1\ldots x+y)$ for all y sensitivity levels) or it could be a short multi-purpose range, with the meaning of the number depending on the location in the vector (e.g. the number 1 could refer to both the type “email” and the sensitivity level “junk”).

Note that this description of HVE mentions that attributes are encoded in a (single) vector. This means that if there are multiple metadata attributes, their individual encodings need to be combined somehow. The easiest way to do this is simply by concatenating the individual representations. So, considering a scheme with two metadata attributes “id” and “date”, where “id” is represented by six elements representing characters and “date” by three elements representing day, month and year, an attribute vector could be of the form $(c_1, c_2, c_3, c_4, c_5, c_6, dd, mm, yy)$. The advantage of this construction is that it ensures that all attributes are taken into account during query evaluation, resulting automatically in conjunctive queries.

Note too, that the description mentions that all metadata and query vectors should be of equal alignment. What this means is that the partitioning of a vector to corresponding attributes should be identical for each metadata and query vector. That is, the location of an attribute’s representation inside the metadata or query vector should be the same throughout the scheme. This is to ensure that queries are correctly interpreted. Of course, by fixing the location of attributes inside a vector, the maximum length of an attribute’s representation is automatically restricted. In some cases, where metadata attributes to be encoded are not of a naturally fixed length (e.g. when encoding variably sized keywords as collections of characters), this may prove to be a problem. In such a case, a default length has to be chosen and individual items have to be padded or truncated to fit this length.

Translating these characteristics into strengths and weaknesses:

Strength HVE schemes have an inherent support for conjunctive queries.

Weakness Metadata and query vectors have to adhere to a strict scheme-specific vector layout

3.1.2 Binary HVE

The expressiveness of HVE lies in the way that simple wildcard queries can be elevated to enable more expressive queries by using creative representations. The most generic way of doing this is through a construction that will be referred to in the rest of this thesis as binary HVE.

The intuition behind a binary HVE scheme is simple: if you want to support some query for a given attribute, then represent that attribute by the collection of outcomes of all such queries.

For instance, suppose there is a metadata attribute describing a person’s family ties and you want to support queries of the type “does this person have a family member of type t". Then you could represent the attribute by a series of yes/no answers (or ones and zeroes), with one answer for every value of t.

So, suppose t can be one of the values {mother, father, brother, sister, son, daughter}, then a person with just a father and a brother could be represented as $(0,1,1,0,0,0)$ and person with both parents and a daughter as $(1,1,0,0,0,1)$. Consequently, queries can be constructed using specific values to select preferred outcomes and wildcards to ignore the rest. That is, adding more specific values to the query acts as a conjunction over that specific attribute: where the query $(\star,1,\star,\star,\star,\star)$ returns all persons with a father, the query $(\star,1,1,0,\star,\star)$ returns only those persons with a father and a brother but no sister.

To put it a bit more formally, a binary HVE scheme can be created for any given *context*, where the context consists of (at least) one:

Metadata attribute — the raw metadata information that the scheme wants to make queryable. The set of all possible values of a metadata attribute is denoted by X.

Universe U — a finite indexed set of size n of possible secondary input values to the query.[1]

Predicate $p : X \times U \to \{0,1\}$ — the Boolean (yes/no) function over the (value of a) metadata attribute and universe that the scheme needs to support.

Given such a context, a binary HVE scheme defines the metadata and query vectors as follows:

Metadata vector — the n dimensional binary vector $\vec{V} := (v_1,\dots,v_n)$ that represents the value x of a metadata attribute by setting $v_i = p(x,u_i)$ for each $u_i \in U$

Query vector — the n dimensional vector $\vec{Q} := (q_1,\dots,q_n)$ that represents a query for specific outcomes r_i of the predicate p on specific secondary input values $t_i \in U$ by setting

$$\text{for each } u_j \in U : \quad q_j = \begin{cases} r_i & \text{if } u_j = t_i \\ \star & \text{otherwise} \end{cases}$$

To illustrate this using the aforementioned example.

Step 1: Defining the context To begin, define the metadata attribute as the description of a person's family relations and set X to be the set of all combinations of the values {mother, father, brother, sister, son, daughter}.

Next, define the predicate that checks whether the metadata attribute contains a specific family member as

$$p(x,u) = u \overset{?}{\in} x$$

and define the universe $U = \{u_1,u_2,u_3,u_4,u_5,u_6\}$ of size $n = 6$ with $u_1 =$ mother, $u_2 =$ father, $u_3 =$ brother, $u_4 =$ sister, $u_5 =$ son and $u_6 =$ daughter.

[1]Note that this is not the same as the alphabet Σ, which in the case of binary HVE is $\{0,1\}$

Step 2: creating vectors Using the predicate p and the universe U of size 6, we can construct the metadata vector $\vec{V} := (v_1, \dots, v_6)$ that represents the family description x as follows:

$$\text{for each } u_i \in U,\ v_i = \begin{cases} 1 & \text{if } u_i \in x \\ 0 & \text{otherwise} \end{cases}$$

To create a query for all records with specific requirements on the family description, first translate that query into specific predicate outcomes. To do this, define two sets T and F, where T contains all family members that *should* be in the description and the set F contains all members that *should not* be in the description according to the query (i.e. $T = \{u_i \,|\, p(x, u_i) \text{ should be } 1\}$ and $F = \{u_i \,|\, p(x, u_i) \text{ should be } 0\}$. Then, construct the query vector $\vec{Q} := (q_1, \dots, q_6)$ as follows:

$$\text{for each } u_i \in U,\ q_i = \begin{cases} 1 & \text{if } u_i \in T \\ 0 & \text{if } u_i \in F \\ \star & \text{otherwise} \end{cases}$$

So, suppose there is a record with a family description that contains a brother, sister, son and daughter (so $x = \{u_3, u_4, u_5, u_6\}$) and a query is performed for all records with a family description that specifically contains a sister (i.e. $T = \{u_4\}, F = \{\}$), this gives the following vectors:

$$\begin{aligned} i = \ & \ \ 1\ \ 2\ \ 3\ \ 4\ \ 5\ \ 6 \\ x = \{u_3, u_4, u_5, u_6\} \rightarrow \vec{V} = & \ (0, 0, 1, 1, 1, 1) \\ T = \{u_4\}, F = \{\} \rightarrow \vec{Q} = & \ (\star, \star, \star, 1, \star, \star) \end{aligned}$$

In this case a match occurs, as every non-wildcard element in the query vector matches the corresponding element in the metadata vector.

Condensing this information into strengths and weaknesses gives:

Strength Binary HVE schemes can support any arbitrary predicate of the form $p : x \times U \rightarrow \{0, 1\}$

Weakness The type of predicates supported in a query are predetermined by the results encoded in the metadata vectors

Weakness Metadata vectors grow linear in each of the universes used by supported predicates

3.1.3 Constructions

Although the previous sections give a good impression of the general capabilities and functionality of HVE, they have not yet dealt with the actual encryption of metadata and queries. More specifically, the previous sections have only discussed the format of the inputs to the *Encrypt* and the *GenerateToken* algorithms without discussing the algorithms themselves. The main reason for this is that, while the original paper about HVE does provide a construction, the concept behind HVE is construction independent. That is, any scheme that supports wildcard queries as described in the previous sections can be rightfully called an HVE.

Of course, this does not mean that there are no similarities between individual HVE constructions that may be analyzed. After compiling a list of HVE schemes that have been devised since the introduction of HVE [BW07, IP08, BIP09, BIP10, LL11, SvLN$^+$10, CIP11] and comparing them, the following high level standard approach can be described for the individual algorithms:

Encrypt

1 Create a ciphertext with for each individual value in the metadata vector $x \geq 1$ elements that hide the respective value in their exponent:
$\vec{V} = (v_1, \dots, v_n) \rightarrow CT = \{C_{1,i}, \dots, C_{x,i}\}_{i=1}^{n}$;

2 Randomize each element:
$CT = \{C_{1,i}, \dots, C_{x,i}\}_{i=1}^{n} \rightarrow CT = \{C_{1,i}R_{1,i}, \dots, C_{x,i}R_{x,i}\}_{i=1}^{n}$;

3 Add $z \geq 1$ additional elements that will help with canceling the randomness:
$CT = \{C_{1,i}R_{1,i}, \dots, C_{x,i}R_{x,i}\}_{i=1}^{n} \rightarrow$
$CT = \{C_{1,i}R_{1,i}, \dots, C_{x,i}R_{x,i}\}_{i=1}^{n}, \{C_{0,1}, \dots, C_{0,z}\}$.

GenerateToken

1 Define the set J of the indexes of non-wildcard values in the query:
$\vec{Q} = (q_1, \dots, q_n) \rightarrow J = \left\{ j \,|\, q_j \neq \star \right\}$;

2 Create a token with for each non-wildcard value in the query vector $x \geq 1$ elements that hide the respective value in their exponent:
$\vec{Q} = (q_1, \dots, q_n) \rightarrow TK = J, \{T_{1,i}, \dots, T_{x,i}\}_{i \in J}$;

3 Randomize each element:
$TK = J, \{T_{1,i}, \dots, T_{x,i}\}_{i \in J} \rightarrow TK = J, \{T_{1,i}R_{1,i}, \dots, T_{x,i}R_{x,i}\}_{i \in J}$;

4 Add $z \geq 1$ additional elements that will help with canceling the randomness:
$TK = J, \{T_{1,i}R_{1,i}, \dots, T_{x,i}R_{x,i}\}_{i \in J} \rightarrow$
$TK = J, \{T_{1,i}R_{1,i}, \dots, T_{x,i}R_{x,i}\}_{i \in J}, \{T_{0,1}, \dots, T_{0,z}\}$.

Test / Decrypt

1 Use the indexes $J = \left\{ j \,|\, q_j \neq \star \right\}$ from the token to find the relevant ciphertext elements:
$CT = \{C_{1,i}R_{1,i}, \dots, C_{x,i}R_{x,i}\}_{i=1}^{n}, \{C_{0,1}, \dots, C_{0,z}\} \rightarrow$
$CT = \{C_{1,i}R_{1,i}, \dots, C_{x,i}R_{x,i}\}_{i \in J}, \{C_{0,1}, \dots, C_{0,z}\}$

2 Use bilinear mappings to combine the different elements of the ciphertext and the token in such a way that the resulting element of the target group evaluates a predetermined function in its exponent.

3 This function should be chosen such that it evaluates to a static value in the case of a match and to random in the case of a non-match.

Three of the studied approaches deviate from this generalized standard approach:

[LL11] deviates slightly by not using individual elements to hide individual non-wildcard values, but by hiding them combined in a single element through multiplication.

[BIP10] deviates more, by appending dummy elements to the token for each wildcard in the query. These dummy elements are treated as if they were created using non-wildcard elements and are necessary to reach the threshold value of the secret sharing scheme that is used as the evaluation function.

[SvLN+10] deviates the most, by basing the ciphertext size on the maximum number of wildcards allowed in a query vector and hiding all of the values of the metadata vector combined in the exponents of individual elements. Similarly, it bases token size on the actual number wildcards used in a query and hides all of the non-wildcard values of the query vector combined in the exponents of individual elements.

When looking at the aforementioned standard approach more closely, two characteristics become apparent. First, the size of a token is proportional to the number of non-wildcard elements in the query vector. This means there is no strict relation between ciphertext size and token size (and with token size, the complexity of the Decrypt / Test algorithm). This is nice, because it allows HVE schemes to use ciphertexts of arbitrarily large sizes without suffering a penalty in efficiency as long as the amount of non-wildcard elements in a query stays low (e.g. as in binary HVE schemes).

However, since the independence of ciphertext size relies on the ability to determine which elements of a ciphertext are relevant for a given query (i.e. by communicating the positions of non-wildcard elements), the structure of the query vector is by definition public.

This means that in the earlier examples, the server knows that a query is for a word of two letters of which the first two are fixed (without knowing what these letters are) or that a query is for documents of all sensitivity levels for some specific (but unknown) type.

In such cases, it may be argued that the leakage is somewhat acceptable. However, in the case of a binary HVE the position of the non-wildcard elements determine the inputs to the predicate, so knowing the location means knowing which yes/no question was asked (just not which answer is expected). To illustrate this, recall the situation used to illustrate binary HVE:

$$\begin{aligned} i = &\; \;1\;\; 2\;\; 3\;\; 4\;\; 5\;\; 6 \\ x = \{u_3, u_4, u_5, u_6\} \rightarrow \vec{V} = &\; (0,0,1,1,1,1) \\ T = \{u_4\}, F = \{\} \rightarrow \vec{Q} = &\; (\star,\star,\star,1,\star,\star) \end{aligned}$$

Here, just knowing that the query only has some value y at index 4 and that $y \in \{0,1\}$ means that the query was either $u_4 \overset{?}{\in} x = 1$ or $u_4 \overset{?}{\in} x = 0$, giving the attacker a fifty-fifty probability of guessing the right query (maybe more if one of the two queries is nonsensical in the given context).

Moreover, because this advantage depends only on the outcome of the query (0 or 1) and not on the actual value in the query, the attacker would still have a fifty-fifty probability of guessing the right query if the universe was $\{u_1, \ldots, u_{10^{42}}\}$ instead of just $\{u_1, \ldots, u_6\}$.

So, condensing these characteristics of the standard approach into strengths and weaknesses:

Strength The Test / Decrypt algorithm is efficient for queries with relatively few non-wildcard elements.

Weakness Tokens leak the structure of their input query vectors (which in the case of binary HVE means most of the query).

Although [BIP10] and [SvLN+10] deviate from the standard, they do both contain the same weakness as the standard approach. By communicating the locations of all wildcard elements, [SvLN$^+$10] automatically exposes the locations of the non-wildcard elements, while in [BIP10] the locations can be found by filtering all indexes in the token that fall within the range of query vector positions.[2]

Because [BIP10] uses the aforementioned padding mechanism, its tokens are always of the same size and proportional to the metadata / query vector length. This means it does not offer the same efficiency advantage as the standard approach.

[SvLN$^+$10] does offer an increased efficiency, but does it in a sort of inversed way when compared with the standard approach, as in [SvLN$^+$10] the token size is proportional to the number of wildcards, instead of to the number of non-wildcards.

3.2 Inner Product Encryption

3.2.1 Basics

Inner Product Encryption (IPE), first proposed in [KSW08], is a response to HVE in the sense that it was created to fill some of the gaps that were left by HVE.

Inner Product Encryption is, just like HVE, a vector based approach, meaning that both metadata and queries are represented by vectors. The way that IPE constructs and uses these vectors, however, is very different from HVE. Depending on the query types that the IPE scheme should support, metadata and query vectors are constructed in such a way that their inner product[3] evaluates to zero in the case of a match.

This means that unlike HVE matching, which is based on a relation between individual elements (i.e. that for every individual element in the query vector it either equals the corresponding element in the metadata vector or the wildcard element), IPE matching does not care about (possible relations between) individual elements of metadata and query vectors, just as long as the matches occur if and only if they satisfy the zero inner product property.

By removing the constraint of a specific relation between individual elements, IPE has opened a path to new imaginative usages of the metadata and query vector.

A simple example of such a new usage is the use of polynomial evaluation to support disjunctions. Suppose that an IPE scheme wants to support queries over a single metadata attribute x, with a maximum of n literals (i.e., queries are of the form "$x = v_1$ or $\dots$ or $x = v_i$" with $1 \leq i \leq n$). Such a scheme would need to evaluate a function that can be represented by an inner product which evaluates to zero for each of the values in $\{v_i \,|\, 1 \leq i \leq n\}$. One function that satisfies that requirement is the function $f(x) = (x - v_1) \cdot \dots \cdot (x - v_i)$. The truth of this can be easily verified:

- The function's roots correspond to $\{v_i\}$, so the function evaluates to zero as soon as $x = v_i$ for some i, $1 \leq i \leq m$.

- The function can be rewritten as a polynomial of degree i: $f(x) = a_0 x^0 + \dots + a_i x^i$, which corresponds to the inner product of the two vectors $\vec{V} = (x^0, \dots, x^i)$ and $\vec{Q} = (a_0, \dots, a_i)$.

Although this shows that disjunctive queries of length i can be achieved for arbitrary values of i, this is not yet enough. This is because inner products can only be calculated

[2]In [BIP10], indexes may be of the range $(1 \dots 2l - 1)$ where l is the length of a metadata / query vector. Here, indexes in the range $(l + 1 \dots 2l - 1)$ correspond to the aforementioned dummy values while indexes in the range $(1 \dots l)$ correspond to actual non-wildcard elements.

[3]The inner product of two vectors $\vec{x}$ and $\vec{y}$ of length n, also written as $\langle \vec{x}, \vec{y} \rangle$, is the total sum of the pair wise multiplication of two vector's elements: $\langle \vec{x}, \vec{y} \rangle = \sum_{i=1}^{n} x_i y_i$.

for vectors of equal length, which means that a scheme with $\vec{V} = (x^0, \ldots, x^i)$ and $\vec{Q} = (a_0, \ldots, a_i)$ implicitly fixes i to some specific value. To overcome this problem, for $i < n$ the vectors $\vec{V}$ and $\vec{Q}$ are extended in the following manner: $\vec{V} = (x^0, \ldots, x^n)$ and $\vec{Q} = (\bar{a}_0, \ldots, \bar{a}_n)$, with $\bar{a}_j = a_j$ for $0 \le j \le i$ and $\bar{a}_j = 0$ for $i < j \le n$. The additional zeroes in the query vector cancel the metadata vector elements with index greater than i, allowing the usage of any $i \le n$.

Additionally, conjunctions can be achieved by concatenation in the same way that concatenation achieved conjunction in HVE. Moreover, as will be seen in the next section, IPE supports arbitrary predicates by offering its own HVE implementation.

This leads to the following strengths and weaknesses:

Strength A broad spectrum of query types, including both conjunctions and disjunctions.

Weakness Fixed metadata and query vector lengths.

3.2.2 HVE in IPE

As briefly alluded to in the previous section, HVE can be implemented inside IPE. [KSW08] shows a straightforward way of doing this by representing elements in an HVE vector by two elements in the corresponding IPE vector. This approach is referred to as the *KSW Approach* throughout this thesis.

In the KSW Approach, metadata vector elements are represented by two elements of the form $(-v_i r_i, r)$, where r_i is some random non-zero value and v_i is the value of the element in position i.

Similarly, non-wildcard query vector elements are represented by two elements of the form $(1, q_i)$.

As can be readily seen, their inner product $\langle(-v_i r_i, r_i), (1, q_i)\rangle = -v_i r_i + r_i q_i = r_i(q_i - v_i) = 0$ if and only if $q_i = v_i$.

To implement the wildcard behaviour, a wildcard element is represented by $(0,0)$, which always results in a zero inner product.

So, to give the formal construction:

Given an HVE scheme with a metadata vector $\vec{V}^{HVE}$ and a query vector $\vec{Q}^{HVE}$, define l as the length of each vector, i.e. $l = \left|\vec{V}^{HVE}\right|$.

Then define the corresponding IPE metadata and query vectors $\vec{V}^{IPE}$ and $\vec{Q}^{IPE}$ as follows:

$$\vec{V}^{IPE} = \left(v_1^{IPE}, \ldots, v_{2l}^{IPE}\right), \text{with for } i = 1 \ldots l:$$
$$v_{2i-1}^{IPE} = -v_i^{HVE} r_i \quad ; \quad v_{2i}^{IPE} = r_i$$
$$r_i \in_R$$

$$\vec{Q}^{IPE} = \left(q_1^{IPE}, \ldots, q_{2l}^{IPE}\right), \text{with for } i = 1 \ldots l:$$
$$q_{2i-1}^{IPE} = \begin{cases} 1 & \text{if } q_i^{HVE} \neq \star \\ 0 & \text{if } q_i^{HVE} = \star \end{cases} \quad ; \quad q_{2i}^{IPE} = \begin{cases} q_i^{HVE} & \text{if } q_i^{HVE} \neq \star \\ 0 & \text{if } q_i^{HVE} = \star \end{cases}$$

As an illustration, consider the following HVE metadata and query vectors:

$$\vec{V}^{HVE} = (5,6,8,9)$$
$$\vec{Q}^{HVE} = (5,\star,8,\star)$$

Then, using a random vector $(r_1,\ldots,r_5) = (3,8,9,2)$ we get

$$\begin{array}{rl}
\vec{V}^{HVE} = (& 5 \quad , \quad 6 \quad , \quad 8 \quad , \quad 9 \quad) \\
\vec{V}^{IPE} = (& -5\cdot 3 \ , \ 3 \ , -6\cdot 8 \ , \ 8 \ , -8\cdot 9 \ , \ 9 \ , -9\cdot 2 \ , \ 2 \ ) \\
= (& -15 \ , \ 3 \ , \ -48 \ , \ 8 \ , \ -72 \ , \ 9 \ , \ -18 \ , \ 2 \ )
\end{array}$$

$$\begin{array}{rl}
\vec{Q}^{HVE} = (& 5 \quad , \quad \star \quad , \quad 8 \quad , \quad \star \quad) \\
\vec{Q}^{IPE} = (& 1 \ , \ 5 \ , \ 0 \ , \ 0 \ , \ 1 \ , \ 8 \ , \ 0 \ , \ 0 \ )
\end{array}$$

$$\begin{aligned}
\left\langle \vec{V}^{IPE}, \vec{Q}^{IPE} \right\rangle &= \left(\begin{array}{cccccccccccccccc} -15 & & 3 & & -48 & & 8 & & -72 & & 9 & & -18 & & 2 \\ \cdot & + & \cdot & + & \cdot & + & \cdot & + & \cdot & + & \cdot & + & \cdot & + & \cdot \\ 1 & & 5 & & 0 & & 0 & & 1 & & 8 & & 0 & & 0 \end{array} \right) \\
&= -15 + 15 + 0 + 0 + -72 + 72 + 0 + 0 \\
&= 0
\end{aligned}$$

Note that the randomness that is introduced by the $\{r_i\}$ is essential to the correctness of the matching function, and not just a way to make encryption probabilistic. Although the aforementioned scheme is correct in the sense that a match occurs whenever it should, it is theoretically possible to get false positives. This is due to the nature of IPE, which only cares about the result of the entire inner product (and not about the pairwise sub inner products). The random numbers $\{r_i\}$ are there to reduce the probability of such false positives to the point where it becomes negligible. As an illustration, consider the HVE vectors $\vec{V}^{HVE} = (2,3)$ and $\vec{Q}^{HVE} = (4,1)$ and a random vector $(54,21)$. Clearly the HVE vectors should not match, and they do not when using the aforementioned random vector. But they would in the special case of "random" vector $(1,1)$):

$$\begin{array}{lrl}
\text{Random vector } (54,21): & \vec{V}^{IPE} = & (-108 \ ,54, -63 \ ,21) \\
& \vec{Q}^{IPE} = & (\ 1 \ ,4 \ ,1 \ ,1) \\
& \left\langle \vec{V}^{IPE}, \vec{Q}^{IPE} \right\rangle = & -108 + 216 - 63 + 21 \\
& = & 66 \\
& \neq & 0 \qquad \text{(true negative)} \\
\text{Special case } (1,1): & \vec{V}^{IPE} = & (\ -2 \ ,1 \ ,-3 \ ,1) \\
& \vec{Q}^{IPE} = & (\ 1 \ ,4 \ ,1 \ ,1) \\
& \left\langle \vec{V}^{IPE}, \vec{Q}^{IPE} \right\rangle = & -2 \ +4 \ -3 \ +1 \\
& = & 0 \qquad \text{(false positive)}
\end{array}$$

As can be readily seen, the lower the impact of individual HVE elements is on the total of the inner product, the higher the probability of a false positive. By picking the r_i from a sufficiently large range, the impact of individual HVE elements can be increased and the probability of false positives can be reduced to an acceptable level.

Finally notice that, because there is no such thing as a wildcard element within IPE, there is no communication of (non-)wildcard indexes. On the positive side, this means that the KSW Approach does not suffer from the inherent information leakage that was found in the HVE schemes mentioned before. On the negative side, this also means that it cannot use index information to reduce the number of necessary computations, as seen in the standard approach to HVE schemes.

Condensing the aforementioned into strengths and weaknesses:

Strength No information leakage through (non-)wildcard indexes.

Weakness Complexity of decryption is always linear in the length of the metadata and query vectors.

Weakness Metadata and query vectors are twice as long as their HVE counterparts.

3.2.3 Constructions

Just as with HVE, IPE as described above only deals with the creation of and relation between metadata and query vectors. And just as with HVE, this has led to the creation of a variety of constructions. After compiling a list of IPE schemes [KSW08, SSW09, LOS$^+$10, SBGN10] and comparing them, the following high level standard approach can be described for the individual algorithms:

Encrypt

1 Create a ciphertext with for each individual value in the metadata vector $x \geq 1$ elements that hide the respective value in their exponent: $\vec{V} = (v_1, \dots, v_n) \rightarrow CT = \{C_{1,i}, \dots, C_{x,i}\}_{i=1}^{n}$;

2 Randomize each element using one or more common randomization factors $CT = \{C_{1,i}, \dots, C_{x,i}\}_{i=1}^{n} \rightarrow CT = \{C_{1,i}R_1, \dots, C_{x,i}R_x\}_{i=1}^{n}$;

GenerateToken

1 Create a token with for each non-wildcard value in the query vector $x \geq 1$ elements that hide the respective value in their exponent: $\vec{Q} = (q_1, \dots, q_n) \rightarrow TK = \{T_{1,i}, \dots, T_{x,i}\}_{i=1}^{n}$;

2 Randomize each element using one or more common randomization factors $TK = \{T_{1,i}, \dots, T_{x,i}\}_{i=1}^{n} \rightarrow TK = \{T_{1,i}R_1, \dots, T_{x,i}R_x\}_{i=1}^{n}$;

Decrypt / Test

1 Use bilinear mappings to combine the different elements of the ciphertext and the token in such a way that the resulting element of the target group evaluates a function in its exponent of the form $r \cdot \langle \vec{v}, \vec{q} \rangle + c$, where r represents a combination of the random factors in the ciphertext and token.

Due to the form of the function, an inner product of zero (i.e. a match) will always cancel out whichever random factors are combined in r and result in some value that is predetermined by c. If that value is known at the time of decryption, it can be used to recognize a correct match. If it is not known at time of decryption but is at the time of encryption, it can be used as an encryption key. Non-matches, on the other hand, will always result in a value that is seemingly random due to the influence of r. The main differences between the aforementioned IPE constructions lie in the familiar three dimensions of security, efficiency and expressiveness:

[SSW09]
Security: selective plaintext and predicate privacy
Efficiency: composite group order of four primes, $2L+2$ token elements.

[LOS$^+$10]
Security: full (non-selective) plaintext privacy.
Efficiency: group order of a single prime, $2L+3$ token elements.

[SBGN10]
Expressiveness: support disjunctions of multiple zero inner products by creating a single monolithic inner product.
Efficiency: new metadata and query vector size $L^{SBG} = L^n$ for disjunction over n IPE vectors of size L.

What is interesting about the high level approach as described here, although it may be a bit of an oversimplification, is that the Encrypt and GenerateToken algorithm seem to be identical. This is because in an inner product, both input vectors are treated the same way functionally and can be switched without affecting the outcome (i.e. the inner product operation is commutative). [SSW09] use this similarity in their symmetric IPE construction to prove predicate privacy and perhaps it may be possible to reuse their approach with other IPE constructions.

As the construction in [SSW09] is currently the only IPE construction with proven predicate privacy, it is chosen as the prime example of symmetric IPE and is from hereon referred to as *the SSW Construction*.

Strength Proven predicate privacy for a specific construction

Strength Potential predicate privacy for future constructions in symmetric settings

Chapter 4

Results

This chapter provides the results of step two of the approach defined in section 1.5, ultimately leading to an answer to the main research question. Following the iterative process of step two, section 4.1 defines the main intuition; section 4.2 details the first attempt at a construction, which is consequently proven to be inherently broken; and section 4.3 details the final construction, which is proven to meet the requirements of the research question. Additionally, section 4.3 provides an alternative to the main construction, that may be more efficient than the main construction in two specific use cases.

4.1 High level intuition

4.1.1 The approach

The main goal of this thesis is to create a searchable encryption scheme that meets the following three requirements:

- the same security guarantees as the SSW Construction;
- greater efficiency than the KSW Approach in conjunction with the SSW Construction;
- at least the same expressiveness as HVE.

After studying the overall strengths and weaknesses of HVE and IPE, it is clear that designing a solution that fits all these requirements is going to be tricky:

- In order for the solution to meet the expressiveness requirement it needs to support wildcard queries, so the solution should allow for an implementation of HVE
- In order for the solution to meet the security requirements it cannot communicate wildcard positions, so the Test / Decrypt algorithm will have to utilize all of the elements in a ciphertext and token
- In order for the solution to meet the efficiency requirement, its token should be smaller than its counterpart in the KSW Approach within the SSW construction.[1]

[1]Assuming that the Test / Decrypt algorithm performs one pairing per element in the token, which holds for all current IPE constructions.

This means that there are broadly three ways of approaching the problem:

1 Find a way to reduce the token size of the SSW construction and keep the KSW approach;

2 Keep the SSW construction, but use an alternative way to implement HVE that reduces metadata / query vector size within IPE;

3 Find a more efficient yet equally secure alternative to IPE.

As the first and third option do not seem feasible within the scope of a Master thesis[2], this leaves the second approach.

4.1.2 The intuition

The second approach imposes several limitations on the choice of representation:

1 Matching has to be based on an inner product(due to IPE)

2 The representations have to be of a fixed length (due to IPE)

3 When representing an HVE input vector, all elements of said vector should influence the result(no information may be discarded)

4 The inner product should somehow provide a mechanism to cancel the effects of HVE metadata values in wildcard positions.

While studying current HVE constructions, one construction attracted specific attention by basing its matching method on a calculation that matches all four requirements. This was the HVE construction of Sedghi e.a. in [SvLN$^+$10].

The main intuition behind [SvLN$^+$10] can be explained in the following steps:

Step 1

Given some vector $\vec{V} = (v_1, \ldots, v_L)$ and a set of wildcard locations J, the following equality holds:

$$\sum_{i=1}^{L} v_i \prod_{j\in J} (i-j) = \sum_{\substack{i=1 \\ i\notin J}}^{L} v_i \prod_{j\in J} (i-j)$$

This holds because whenever index i equals a wildcard location j in J, the corresponding value v_i gets canceled as the product $\prod_{j\in J}(i-j)$ evaluates to zero. This means that the summation $\sum_{i=1}^{L}$ is only influenced by values of i that are not in the set of wildcard locations, so it can be rewritten as $\sum_{\substack{i=1 \\ i\notin J}}^{L}$.

Step 2

The above equality can be used to detect whether two vectors $\vec{V} = (v_1, ;v_L)$ and $\vec{Q} = (q_1, ;q_L)$ agree on all elements in non-wildcard positions:

$$\sum_{i=1}^{L} v_i \prod_{j\in J} (i-j) = \sum_{\substack{i=1 \\ i\notin J}}^{L} q_i \prod_{j\in J} (i-j)$$

$$\text{if } v_i = q_i \text{ for all } i \notin J$$

[2]These are popular open questions that have already received quite a bit of attention from people with a lot more resources to their disposal.

Step 3

The product $\prod_{j\in J}(i-j)$ can be rewritten as a polynomial of the form $\sum_{k=0}^{n} a_k i^k$, where n is the number of wildcard locations. This means the left hand side of the equality can be rewritten:

$$\sum_{i=1}^{L} v_i \prod_{j\in J}(i-j) \to \sum_{i=1}^{L} v_i \sum_{k=0}^{n} a_k i^k \to \sum_{k=0}^{n} a_k \sum_{i=1}^{L} v_i i^k$$

Moreover, the coefficients a_k of the polynomial are completely determined by J and can be calculated using one of Viète's formulas.[3]

Step 4

As the right hand side can be calculated at the time of token generation, $\sum_{i=1}^{L} v_i i^k$ can be calculated for every possible value of k at the time of ciphertext generation and the a_k can be calculated at the time of decryption as long as J is known, the equality can also be tested when the values are hidden in exponents and will only need n elements in the ciphertext.

The interesting bit here is step 3, because it shows that the left hand side of the equation has a form which can be rewritten as an inner product:

$$\sum_{k=0}^{n} a_k \sum_{i=1}^{L} v_i i^k = \langle \vec{x}, \vec{y} \rangle \text{, with}$$

$$\begin{aligned} \vec{x} &= (x_0,\ldots,x_n), & x_i &= \sum_{j=1}^{L} v_j j^i, & &\text{for } 0 \le i \le n \\ \vec{y} &= (y_0,\ldots,y_n), & y_i &= a_i, & &\text{for } 0 \le i \le n \end{aligned}$$

This means that the calculation fits within the first limitation.

Moreover, if we define a maximum number of wildcards $N \le L$, this is equivalent to the inner product $\langle \vec{x}, \vec{y} \rangle$, with

$$\begin{aligned} \vec{x} &= (x_0,\ldots,x_N), & x_i &= \sum_{j=1}^{L} v_j j^i, \text{ for } 0 \le i \le N \\ \vec{y} &= (y_0,\ldots,y_N), & y_i &= \begin{cases} a_i & \text{for } 0 \le i \le n \\ 0 & \text{for } m < i \le N \end{cases} \end{aligned}$$

which results in a fixed vector length of $N+1$, fitting the second limitation.

Step 2 shows that the calculation also fits the third and fourth limitation.

Although it may sound like we are done, this is not yet the case. Because even though the above shows how the left hand side of the equation can be made into an inner product, this inner product does not evaluate to zero upon a match. Instead, it evaluates to a specific value t that depends on the query.

The problem of compensating for this t is left open for actual constructions.

[3]The formula will not be discussed here in detail. For more information, read the paper [SvLN+10].

4.1.3 Efficiency, expressiveness and security

Note that if the problem of compensating for t can be solved, the aforementioned intuition results in an HVE implementation that is at least almost twice as efficient as the KSW Approach as the input to the underlying IPE scheme would be of size $N+1$ instead of $2L$, with $N \leq L$. This would instantly satisfy the *efficiency* requirement.

Similarly, as it would effectively be an HVE scheme, the *expressiveness* requirement would be satisfied too.

Whether the *security* is satisfied too depends heavily on the way the t is compensated. If the compensation does not change the SSW Construction, its security can be inherited. Otherwise, an additional security proof is needed.

4.2 First attempt

4.2.1 Intuition

The first attempt at compensating for the value t revolves around adapting the SSW Construction to allow for specific-value non-zero inner product matching, the idea being that a generic adaptation to non-zero inner products could be reused in future constructions.

The first step in adapting the SSW Construction to allow for specific-value non-zero inner product matching, is to analyze how the SSW Construction works.

In the SSW Construction ciphertext object are of the form

$$CT = \left(\begin{array}{c} C = S \cdot g_p^y, \quad C_0 = S_0 \cdot g_p^z \\ \left\{ C_{1,i} = h1,i^y \cdot u_{1,i}^z \cdot g_q^{\alpha v_i} \cdot R_{1,i}, \quad C_{2,i} = h_{2,i}^y \cdot u_{2,i}^z \cdot g_q^{\beta v_i} \cdot R_{2,i} \right\}_{i=1}^n \end{array} \right)$$

and tokens are of the form

$$TK = \left(\begin{array}{c} K = R \cdot \prod_{i=1}^n h_{1,i}^{-r_{1,i}} \cdot h_{2,i}^{-r_{2,i}}, \quad K_0 = R_0 \cdot \prod_{i=1}^n u_{1,i}^{-r_{1,i}} \cdot u_{2,i}^{-r_{2,i}} \\ \left\{ K_{1,i} = g_p^{r_{1,i}} \cdot g_q^{f_1 q_i} \cdot S_{1,i}, \quad K_{2,i} = g_p^{r_{2,i}} \cdot g_q^{f_2 q_i} \cdot S_{2,i} \right\}_{i=1}^n \end{array} \right)$$

where the v_i represent values from a metadata vector, the q_i represent values from a query vector, all other exponents are freshly picked random integers, $h_{i,j}$ and $u_{i,j}$ are fixed elements from G_p and all capital letters X represent fresh random values from subgroup G_x.

The SSW Decrypt / Test algorithm consists of the following test:

$$e(C,K) \cdot e(C_0,K_0) \cdot \prod_{i=1}^{n} e(C_{1,i},K_{1,i}) \cdot e(C_{2,i},K_{2,i}) \stackrel{?}{=} \varepsilon_T$$

which basically does the following:

- cancel out the various masking elements from subgroups G_r and G_s;
- perform a function over the elements in G_p that will evaluate to the identity element if the ciphertext and token are well-formed (whether the correct h and u elements are used, etc.);
- perform a function over the elements in G_q that will evaluate to the identity element if the inner product of metadata and query vector evaluates to zero.

This last part consists of calculating the element

$$g_{Tq}^{(\alpha f_1+\beta f_2)\langle\vec{v},\vec{q}\rangle}$$

where α and β are random exponents from the ciphertext, f_1 and f_2 are random exponents from the token and g_{Tq} is the generator $e(g_q, g_q)$ of subgroup G_{Tq} in the target group.

As can be readily seen, if the inner product $\langle\vec{v},\vec{q}\rangle$ evaluates to zero, then so does the total of the exponent, resulting in the identity element of the target group.

The intuition behind the first attempt is to add a few elements to both the ciphertext and the token that will allow the algorithm to evaluate the function in two separate ways and base a match on whether the result of these two evaluations are equal. This new evaluation is based on the following equality:

$$(\alpha f_1+\beta f_2)\langle\vec{v},\vec{q}\rangle = \alpha f_1\langle\vec{v},\vec{q}\rangle + \beta f_2\langle\vec{v},\vec{q}\rangle$$

Because the intended result of $\langle\vec{v},\vec{q}\rangle$ can be calculated during token generation and fixed to some value π, it is possible to split the information of the new form into ciphertext parts c_i and token parts t_i:

$$\begin{gathered}\alpha f_1\langle\vec{v},\vec{q}\rangle + \beta f_2\langle\vec{v},\vec{q}\rangle \\ \alpha(f_1\langle\vec{v},\vec{q}\rangle) + \beta(f_2\langle\vec{v},\vec{q}\rangle) \\ \alpha(f_1\pi) + \beta(f_2\pi) \\ c_1(t_1) + c_2(t_2)\end{gathered}$$

Because single multiplication in the exponent can be done using a pairing and addition in the exponent can be done using multiplication, the function can be evaluated in exponents:

$$\begin{gathered}g_{Tq}^{\alpha f_1\pi+\beta f_2\pi} \\ g_{Tq}^{\alpha f_1\pi}\cdot g_{Tq}^{\beta f_2\pi} \\ e(g_q,g_q)^{\alpha f_1\pi}\cdot e(g_q,g_q)^{\beta f_2\pi} \\ e(g_q^{\alpha},g^{f_1\pi})\cdot e(g_q^{\beta},g_q^{f_2\pi}) \\ e(c_1,t_1)\cdot e(c_2,t_2)\end{gathered}$$

So, with two extra ciphertext elements and two extra token elements, it should be possible to create a functional scheme for non-zero inner products.

4.2.2 Construction

Given the aforementioned intuition, a construction can be made as follows:

Setup: Identical to SSW

Encrypt: Generate SSW ciphertext $CT = (C, C_0, \{C_{1,i}, C_{2,i}\}_{i=1}^{n})$, return:

$$(CT, C_3 = g_q^{\alpha}, C_4 = g_q^{\beta})$$

GenerateToken: Generate SSW ciphertext $T = (K, K_0, \{K_{1,i}, K_{2,i}\}_{i=1}^n)$, return:

$$(T, K_3 = g_q^{f_1 \pi}, K_4 = g_q^{f_2 \pi})$$

Decrypt / Test: Return:

$$e(C,K) \cdot e(C_0,K_0) \cdot \prod_{i=1}^{n} e(C_{1,i},K_{1,i}) \cdot e(C_{2,i},K_{2,i}) \cdot (e(C_3,K_3) \cdot (C_4,K_4))^{-1} \overset{?}{=} \varepsilon_T$$

4.2.3 Security

Although the functionality of the scheme can be shown in a straightforward manner, the security of such a scheme is a completely different matter.

Two different approaches were taken to proving the security and both of them have failed. The first approach consisted of following the security proof of the SSW Construction. This seemed like a logical choice as the scheme is based on SSW. When this proved to be impossible, the second approach consisted of testing the construction against Katz, Sahai and Waters' "Master Theorem" for hardness in composite order bilinear groups. Although passing the test would not result in a security guarantee for concrete group implementation, it would offer some plausibility to the scheme's hardness. Furthermore, if the scheme could be shown to be easy in the generic group model, then it would be provably broken.

Following the security proof of the SSW Construction.

The first step in following the security proof of the SSW Construction is reproducing the proof of plaintext privacy.

The SSW Construction proves its plaintext privacy by showing that breaking their scheme implies breaking the original IPE construction by Katz, Sahai and Waters [KSW08] (from hereon referred to as the KSW Construction). To do this, they show that they can extend the ciphertexts and tokens generated by the KSW Construction to their respective counterparts in a scheme they call SchemeQ, which is indistinguishable from their main construction, using nothing more than the information in KSW's public key. This way, if there exists an attacker which has an advantage of attacking the SSW Construction (and hence SchemeQ), they can use that attacker to attack the KSW Construction. Without going into too much detail on the way SchemeQ is constructed, the reason that the KSW elements can be extended is due to availability of both the generator g_p of G_p and an element g_qR, where g_q is the generator of group G_q and R a random element from G_r.

In order to extend the KSW ciphertexts and tokens further and go from SchemeQ to the first attempt, we need to add elements that hide information in the exponent of g_q. These elements may be masked by any number of other elements, as long as these masks can be removed using pairings.

When inspecting the public key of the KSW construction,

$$PK = (g_p, g_r, g_qR, \{H_{1,i}, H_{2,i}\})$$

where all $H_{j,i}$ are of the form $g_p^{r_{j,i}} R_{j,i}$, with $R_{j,i}$ random elements in G_r, it is easy to see there is a problem: the only element that exposes g_q in some way is g_qR. This is bad, because it means that we cannot create elements purely of the form g_q^a and, even though we may be able to create masked elements of the form g_q^aR, we do not have access to any elements that may remove the masking. That is, there are no elements of the form $g_q X$ where X is not in G_r, so for any element b that can be created we have that either b does not contain a factor g_q or b contains a factor from G_r. This means that for any pairing $e(g_q^a R, b)$, the result will either have the information in a canceled as $e(g_q^a, b) = \varepsilon_T$ or be indistinguishable from random as $e(g_q^a R, g_q^x R_2) = g_{Tq}^{ax} R_T$, where R_T is random in G_{Tr}.

This means that we cannot recreate the first attempt from the public parameters of KSW. Therefore we cannot recreate the security proof of SSW and we will need to create a security proof from scratch, possibly having to invent new assumptions along the way.

As creation of a novel security proof will never fit within the time frame of a Master's thesis, this approach is abandoned.

Note that this does not yet mean that the scheme is therefore insecure. Especially since the scheme is supposed to run in a symmetric key setting, with no public components to abuse, it could still be that the new scheme is secure (especially if the new elements are masked with elements from G_r and G_s).

Testing the scheme against the "Master Theorem" of hardness.

The second attempt at proving the security tries to deduce the likelihood of the security by testing it against Katz, Sahai and Waters' "Master Theorem" for hardness in composite order bilinear groups [KSW08]. If the scheme passes this test, then it should be hard to break within the generic group model as long as the problem of factorizing the group order is hard.

The idea behind the "Master Theorem" is fairly simple: if two elements are independent of all other public elements[4] and if all pairings using (one of) these two elements are either independent of all other possible pairings with public elements or the same for both elements, then it is very hard to distinguish between them in a decisional problem.

To be a bit more precise, we make use of Theorem A.2 from [KSW08], which states that the decisional problem is hard if

> ...each of T_0 and T_1 is independent of $\{Ai\}$, and furthermore that for all $k \in S$ it holds that $e(T_0, A_k)$ is independent of $\{B_i\} \cup \{e(A_i, A_j)\} \cup \{e(T_0, A_i)\}_{i \neq k}$, and $e(T_1, A_k)$ is independent of $\{B_i\} \cup \{e(A_i, A_j)\} \cup \{e(T_1, A_i)\}_{i \neq k}$

where

- A_i represents all public elements from G;
- T_0 and T_1 are the test elements from G;
- B_i represents all public elements from G_T;
- S is the collection of indexes i for which $e(T_0, A_i) \neq e(T_1, A_i)$.

The following iterative approach is used in testing the first attempt against the master theorem:

1 Set $\{A_i\}$ to contain all ciphertext and token elements of the regular SSW Construction;

[4]Here, independence means that they cannot be reproduced using a linear combination of the other elements

2 Set T_0 to the next element to be added to either ciphertext or token and T_1 to a random element from G;

3 Test the independence requirements posed by the theorem;

  a If the requirements hold:

    I If there are more elements to add, add T_0 to $\{A_i\}$, and return to 2;

    II If there are no more elements to add, the construction has passed the test

  b If the requirements do not hold:

    I If this can be fixed by changing masking method, change the construction and return to 1;

    II If this cannot be fixed, describe a possible attack and accept failure.

Following this approach, it turns out that the addition of *any one* element of the form QX, where Q is an element in G_q and X is an element of some G_x that gets canceled when paired with either ciphertext elements $C_{i,j}$ (or token elements $K_{i,j}$), will break the plaintext (or predicate) privacy of the SSW Construction.

A possible attack using an element QS:
Given:

- two known metadata vectors $\vec{X} = (x_0, \ldots, x_L), \vec{Y} = (y_0, \ldots, y_L)$, for which an attacker knows for some i and j that $x_i/x_j = z \bmod q$ and $x_i/x_j \neq y_i/y_j \bmod q$
- SSW ciphertext $CT = (C, C_0, \{C_{1,k}, C_{2,i}\}_{k=1}^{n})$, which may be of either $\vec{X}$ or $\vec{Y}$
- some element T of the form QS

Assume that CT is made using $\vec{X}$ and test that $e(C_{1,i}, T) = e(C_{1,j}, T^z)$. This should hold because:

$$\begin{aligned}
e(C_{1,j}, T^z) &= e(C_{1,j}, QS^z) \\
&= e(g_q^{\alpha x_j}, Q^z) \\
&= e(g_q^{\alpha}, Q)^{z x_j} \\
&= e(g_q^{\alpha}, Q)^{x_i} \\
&= e(g_q^{\alpha x_i}, Q) \\
&= e(C_{1,i}, T)
\end{aligned}$$

If this equality doesn't hold, the CT was made using $\vec{Y}$.

In the SSW Construction, this leaves only elements that contain a factor QP as elements with a factor from G_q that could be added safely.

As the usage of random elements from G_p would make de-masking impossible, this means that G_p factors would have to be specially constructed in such a way that they cancel each other during the test algorithm. However, if this is done in a way that also allows an attacker to cancel the G_p factor in other ciphertext or token elements, the scheme is broken once again.

Given that the only operations available (i.e. the regular group operation and the pairing) either result in multiplication or addition of exponents and the final result should cancel the effect of the masking element, the mechanism will need to use either $g_{Tp}^{p_1} g_{Tp}^{p_2} = g_{Tp}^0$ or $e(g_p^{p_1}, g_p^{p_2}) = g_{Tp}^0$. Given that the latter will only happen for $p_1 = 0$ or $p_2 = 0$, which are not allowed (as it would effectively return the construction to one which we already know to be broken), the construction would need to use $g_{Tp}^{p_1} g_{Tp}^{p_2} = g_{Tp}^0$.

Recalling the format of the first attempt, $e(c_1,t_1) \cdot e(c_2,t_2)$, both p_1 and p_2 would be of the form $p_{ci} p_{ti}$, with $p_{c1} p_{t1} + p_{c2} p_{t2} = 0$. This leaves two possibilities:

1 all exponents are static

2 at least one of the exponents is chosen at random

If the exponents are static, then the scheme can be broken using any combination of two ciphertexts or tokens, as two instances can be used to compute an element of G_q.

For instance, using a static p_{c1}:

$$g_p^{p_{c1}} Q_1 / g_p^{p_{c1}} Q_2 = Q_1/Q_2 = Q_3 \in G_q$$

If one of the exponents is chosen at random, then the other values of the have to be picked in such a way that they support arbitrary values.

Supposing that p_{t1} is chosen at random, designated as r, this means that:

$$\begin{aligned} p_{c1} r + p_{c2} p_{t2} &= 0 \\ p_{c2} p_{t2} &= -p_{c1} r \\ p_{c2} &= -p_{c1} r / p_{t2} \end{aligned}$$

The only way to guarantee that this equation holds is to pick $p_{t2} = r$, resulting in $p_{c2} = -p_{c1}$.

Sadly, such a construction allows an attacker to cancel the G_p factor of a ciphertext or token and enables a similar attack as before. Given some query vector $\vec{Q} = \{q_0, \ldots, q_L\}$, $T_1 = g_p^{p_{c1}} Q_1$, $T_2 = g_p^{p_{c2}} Q_2$, with $q_i/q_j = x$ (like before), it holds that:

$$\begin{aligned} e(K_{1,j}, T_1^x) \cdot e(K_{1,j}, T_2^x) &= e(g_p^{r_j} g_q^{f_1 q_j}, g_p^{x p_{c1}} Q_1^x) \cdot e(g_p^{r_j} g_q^{f_1 q_j}, g_p^{x p_{c2}} Q_2^x) \\ &= e(g_p^{r_j}, g_p^{x p_{c1}}) \cdot e(g_p^{r_j}, g_p^{x p_{c2}}) \cdot e(g_q^{f_1 q_j}, Q_1^x) \cdot e(g_q^{f_1 q_j}, Q_2^x) \\ &= e(g_p^{r_j}, g_p^{x p_{c1}}) \cdot e(g_p^{r_j}, g_p^{-x p_{c1}}) \cdot e(g_q^{f_1 q_j}, Q_1^x) \cdot e(g_q^{f_1 q_j}, Q_2^x) \\ &= e(g_q^{f_1 q_j}, Q_1^x) \cdot e(g_q^{f_1 q_j}, Q_2^x) \\ &= e(g_q^{f_1 q_j}, g_q^{x z_1}) \cdot e(g_q^{f_1 q_j}, g_q^{x z_2}) \\ &= e(g_q^{f_1}, g_q)^{x q_j z_1} \cdot e(g_q^{f_1}, g_q)^{x q_j z_2} \\ &= e(g_q^{f_1}, g_q)^{x q_j (z_1 + z_2)} \\ &= e(g_q^{f_1}, g_q^{z_1 + z_2})^{x q_j} \\ &= e(g_q^{f_1}, g_q^{z_1 + z_2})^{q_i} \\ &= e(K_{1,i}, T_1) \cdot e(K_{1,i}, T_2) \end{aligned}$$

4.2.4 Result

Although there are various ways to functionally extend the SSW construction in such a way that it could support non-zero inner products, there is no way of doing so without compromising the security of the scheme. This is because any such extension would require the usage of elements that allow for a new calculation in the exponent of g_q and it can be shown using Katz, Sahai and Waters' Master Theorem of hardness that any combination of elements that allow such a calculation can be used to detect linear relations in the plaintext metadata or query vector.

This result can be generalized to serve as a warning for creators of new schemes within a bilinear setting:

Any scheme

- in a bilinear setting of (possibly composite) order $n = p_1 \cdot \ldots \cdot p_p$,
- which hides secret information X in at least two elements C_1, C_2 of the format $C_i = g_x^{x_i} R$, where each x_i is an exponent depending on X, g_x is the generator of common subgroup G_x and R is an element of a different subgroup G_m of (possibly composite) order m,
- in such a way that depending on X there may be a linear relation between the exponents x_i

may leak said linear relation if it exposes any combination of elements T_j

- that can be used to create a pairing with both C_i such that the results r_i are in the target subgroup G_{Tx}.

This includes all elements of the format XL, where $X \in G_x$ and $L \in G_l$, such that l does not share any prime factors with m.[5]

4.3 Main construction

4.3.1 Intuition

The intuition behind the main construction differs from the first in the fact that it is not focused at changing the SSW Construction. Instead, it focuses on the main intuition itself and changes it such that it produces a compatible zero inner product.

The change is based on the realization from the first attempt that the intended inner product can be calculated at the time of token generation. Because the intended inner product can be during token generation, it can be appended to the input query vector. Then, if metadata vectors are extended such that the last element of the query vector is deducted, the result is once more a zero inner product.

That is, given a metadata vector $\vec{V} = (v_0, \ldots, v_n)$ and a query vector $\vec{Q} = (q_0, \ldots, q_n)$

$$\begin{array}{llll} \text{if} & \langle (v_0, \ldots, v_n), (q_0, \ldots, q_n) \rangle & = v_0 q_0 + \cdots + v_n q_n & = \pi \\ \text{then} & \langle (v_0, \ldots, v_n, -1), (q_0, \ldots, q_n, \pi) \rangle & = v_0 q_0 + \cdots + v_n q_n + (-1\pi) & = 0. \end{array}$$

Note that this notation is very generic, which means that any specific-value (π-) inner product scheme can be transformed to a corresponding zero inner product scheme as long as the specific value can be determined at token generation.

[5]Note that this description also covers the KSW Construction, which encodes the query vector in elements of the form $g_q^{x_i} P$ and includes an element $g_q R$ in its public key, which can be paired with these elements to produce elements of target subgroup G_{Tq}. Although this does not necessarily break the KSW Construction, as KSW make no claims regarding predicate privacy, it is interesting to see that the KSW Construction is open to more attacks than the trivial ciphertext generation attack of [ABC+08].

In the case of our main intuition, the π corresponds with

$$\sum_{\substack{i=1 \\ i \notin J}}^{L} q_i \prod_{j \in J} (i-j) \,.$$

4.3.2 Construction

Using this new intuition, the following generic construction can be defined:

Setup(HVE Alphabet Σ, HVE vector length L, Number of wildcards N)

- Setup a plaintext and predicate private IPE scheme IPE for vectors of length $N+2$ and alphabet Σ
- Publish public components of the IPE scheme

Encrypt(HVE metadata vector $\vec{V}^{HVE}$)

- Calculate all $v_k = \sum_{i=1}^{L} v_i^{HVE} i^k$ for $k = 0 \ldots N$
- Set $\vec{V}^{IPE} = (v_0, \ldots, v_N, -1)$
- Return ciphert $C = IPE.Encrypt(\vec{V}^{IPE})$

GenerateToken(HVE query vector $\vec{Q}^{HVE}$)

- Create the set J of all k wildcard positions in $\vec{Q}^{HVE}$
- Use Viète's formula to calculate the coefficients a_i for $i = 0 \ldots k$
- Calculate $\pi = \sum_{\substack{i=1 \\ i \notin J}}^{L} q_i^{HVE} \prod_{j \in J} (i-j)$
- Set $\vec{Q}^{IPE} = (a_0, \ldots, a_k, 0_{k+1}, \ldots, 0_N, \pi)$
- Return token $T = IPE.GenerateToken(\vec{Q}^{IPE})$

Decrypt / Test(Ciphertext C, Token T)

- Return result $R = IPE.Decrypt(C, T)$

Although the SSW Construction is currently the only IPE construction offering both plaintext and predicate privacy, this generic definition allows the usage of any future scheme that support IPE.

4.3.3 Security

In order to prove the construction's security, both the correctness and the privacy requirements need to be met.

Correctness of the scheme depends on both the correctness of the inner product that is used for matching and the correctness of the encryption scheme. The former can be derived directly from the correctness of [SvLN$^+$10], the latter can be derived directly from the correctness of the Decrypt / Test algorithm of the underlying IPE scheme.

Plaintext and predicate privacy are directly determined by the Encrypt and GenerateToken algorithms of the underlying IPE scheme. This is because the scheme is purely focused on preprocessing inputs to the IPE and the result of the preprocessing is a valid inner product matching algorithm. As there exists at least one IPE construction that provides both plaintext and predicate privacy (the SSW Construction), the setup algorithm ensures plaintext and predicate privacy for the entire construction.

As the solution is correct and provides plaintext and predicate privacy, it satisfies the security requirement of the research question.

4.3.4 Efficiency

By extending the input vectors with an additional element, the total length of the input vectors becomes $N+2$, where N is the maximum number of allowed wildcard characters. This is shorter than the $2L$, where L is the length of an HVE input vector, demanded by the KSW Approach for all $L > 2$ as $N \leq L$.

If L is sufficiently large, then in the worst case with $N \approx L$ the vector length is roughly halved, as $(N+2)/2L = (L+2)/2L \approx L/2L = 1/2$. More generally, if L is chosen large enough to make the $+2$ negligible and N is chosen as a specific percentage p of L ($N = Lp$), then the vector length in this construction is only $Lp/2L = p/2$ percent of the vector length in the KSW Approach. In binary HVE schemes, the percentage p will either approach or equal 100%, as searches for a single specific outcome probably need to be supported (which requires $L-1 \approx L$ wildcards). In non-binary HVE schemes, the percentage may be chosen lower in a trade-off between efficiency and expressiveness.

As the solution provides significantly shorter IPE input vectors than the KSW Approach even for $N = L$, and the number of pairings needed for a single test is linear in the length of the IPE input vectors, the solution is said to meet the efficiency requirement.

4.3.5 Expressiveness

As the solution creates a correct HVE implementation within IPE, it offers the same expressiveness as regular HVe. Therefore, the solution also meets the expressiveness requirement.

4.3.6 Result

This solution offers a way to preprocess HVE input vectors in such a way that they can be used within IPE constructions. By choosing an IPE construction with proven plaintext and predicate privacy, this can be used to provide the same plaintext and predicate privacy for HVE schemes. For HVE schemes with vector length L and a maximum number of wildcards $N = Lp \leq L$, the resulting IPE input vectors are of length $Lp+2$, which is significantly shorter than the standard KSW Approach vector length of $2L$.

4.4 Alternative construction

4.4.1 Intuition

After formulating the main construction, we realized that the same technique used in compensating for π could be used to modify original KSW approach such that it produces vectors of length $L+1$ instead of $2L$.

In the original KSW approach, every element in an HVE vector is represented by two elements in the corresponding IPE vector. Metadata elements v_i are represented by a tuple containing its multiplication with a fresh random value r_i and the random value itself: $v_i \rightarrow (-v_i r_i, r_i)$. Query values q_i are represented by either the tuple $(0,0)$ if q_i is a wildcard or by the tuple $(1, q_i)$ if it is not.

The r_i in this construction are used to spread the values inside the metadata vector to reduce the probability of false positives (see also section 3.2.2). However, to achieve such a spread these r_i do not necessarily have to be picked fresh.

If the r_i are fixed for the scheme, then the same trick that is used in the main construction can be used to reduce the size of the IPE vector.:

Original KSW:

$$\vec{V}^{IPE} = \left(v_1^{IPE}, \ldots, v_{2L}^{IPE}\right), \text{with for } i = 1 \ldots L\text{:}$$
$$v_{2i-1}^{IPE} = -v_i^{HVE} r_i \qquad v_{2i}^{IPE} = r_i$$
$$r_i \in_R$$

$$\vec{Q}^{IPE} = \left(q_1^{IPE}, \ldots, q_{2L}^{IPE}\right), \text{with for } i = 1 \ldots L\text{:}$$
$$q_{2i-1}^{IPE} = \begin{cases} 1 & \text{if } q_i^{HVE} \neq \star \\ 0 & \text{if } q_i^{HVE} = \star \end{cases} \qquad q_{2i}^{IPE} = \begin{cases} q_i^{HVE} & \text{if } q_i^{HVE} \neq \star \\ 0 & \text{if } q_i^{HVE} = \star \end{cases}$$

Using a fixed set of r_i:

$$\vec{V}^{IPE} = \left(v_1^{IPE}, \ldots, v_L^{IPE}, -1\right), \text{with for } i = 1 \ldots L, v_i^{IPE} = v_i^{HVE} r_i$$

$$\vec{Q}^{IPE} = \left(q_1^{IPE}, \ldots, q_L^{IPE}, \pi\right), \text{with } \pi = \sum_{i=1}^{l} (q_i r_i)^2, \text{and for } i = 1 \ldots L, \text{:}$$
$$q_i^{IPE} = \begin{cases} q_i^{HVE} r_i & \text{if } q_i^{HVE} \neq \star \\ 0 & \text{if } q_i^{HVE} = \star \end{cases}$$

The only disadvantage here is that this approach does not allow v_i with a value of zero. In the original KSW Appraoch, a zero v_i representation would include the standalone r_i, which assures that searches for non-zero values would not result in a zero inner product. In this reduced form, however, a zero v_i would always result in a zero inner product for the element in said location. Therefore, the alphabet Σ of the HVE to be implemented should be mapped to an alphabet Σ^* that does not contain a zero value. However, as this is a trivial operation, this is not considered an obstacle.

4.4.2 Construction

Applying this intuition gives the following construction:

Setup(HVE Alphabet Σ, HVE vector length L, Number of wildcards N)

- Map Σ to an alphabet Σ^* that does not contain a zero value
- Define r_i for $1 \leq i \leq L$
- Setup a plaintext and predicate private IPE scheme IPE for vectors of length $L+1$ and alphabet Σ^*
- Publish public components of the IPE scheme

Encrypt(HVE metadata vector $\vec{V}^{HVE}$)

- Substitute all v_i in $\vec{V}^{HVE}$ with their counterparts w_i from Σ^*
- Set $\vec{V}^{IPE} = \left(v_1^{IPE}, \ldots, v_L^{IPE}, -1\right)$, with for $i = 1 \ldots L$, $v_i^{IPE} = w_i r_i$
- Return ciphert $C = IPE.Encrypt(\vec{V}^{IPE})$

GenerateToken(HVE query vector $\vec{Q}^{HVE}$)

- Substitute all non-wildcard q_i in $\vec{Q}^{HVE}$ with their counterparts w_i from Σ^*
- Calculate $\pi = \sum_{i=1}^{l} (w_i r_i)^2$
- Set $\vec{Q}^{IPE} = \left(q_1^{IPE}, \ldots, q_L^{IPE}, \pi\right)$, with for $i = 1 \ldots L$, $q_i^{IPE} = 0$ if $q_i = \star$ and $q_i^{IPE} = w_i r_i$ otherwise
- Return token $T = IPE.GenerateToken(\vec{Q}^{IPE})$

Decrypt / Test(Ciphertext C, Token T)

- Return result $R = IPE.Decrypt(C, T)$

4.4.3 Security and Expressiveness

Just like with the main construction, the plaintext and predicate privacy of this construction are fully determined by the plaintext and predicate privacy of the chosen IPE scheme. As there is at least one IPE scheme that provides plaintext and predicate privacy, this construction satisfies the security requirement.

The expressiveness requirement is satisfied as the construction implements HVE.

4.4.4 Efficiency

In this construction, HVE input vectors of length L are transformed into vectors of length $L+1$. Although this means it is not necessarily more efficient than our main construction, it is more efficient than the KSW Approach for all $L > 1$.

When comparing the alternative construction with our main construction, there are two specific cases in which it is more efficient. These are:

1 The case that the maximum number of wildcards equals the total number of elements in a query. In this case, the main construction produces vectors of size $L+2$ while the alternative construction produces vectors of size $L+1$

2 The case that the cost of computing Viètes formula and the expected result π exceeds the cost of the pairings that are avoided due to decreased vector size.

4.4.5 Result

The alternative construction offers an HVE scheme with roughly half the vector length of the KSW Approach and without the computation of Viète’s formula. This makes it a viable candidate for schemes with no restriction on the number of wildcards, such as binary HVE schemes.

Chapter 5

Conclusion

Regarding the question whether it is possible to construct an encryption scheme that is more efficient than the KSW approach within the SSW construction while retaining its security and expressiveness, the answer is yes.

Just like the KSW approach, the two HVE constructions proposed in this thesis preprocess HVE input vectors in such a way that they can be used within a secure IPE construction. The efficiency gain lies in the length of the IPE vectors that the constructions produce. For any HVE scheme that uses input vectors of length L, The KSW Approach produces IPE vectors of length $2L$.

The main construction of this thesis is based on the work of Sedghi e.a. and provides IPE vectors of a length $N+2$, where N is the maximum number of wildcard characters in an HVE query vector. As N is by definition no more than the full length of an HVE input vector, this means vector lengths of at most $L+2$, which is roughly half the vector length of the KSW Approach. This worst case scenario is most likely to occur for binary HVE schemes, where query vectors mostly consist of wildcards. For non-binary HVE schemes, however, this dependence on the maximum number of wildcards can be used to create a suitable balance between expressiveness and the efficiency on a scheme to scheme basis.

The alternative construction of this thesis is based on the work of Katz, Sahai and Waters themselves and uses one of the tricks of the main construction to create IPE vectors of length $L+1$. Although the alternative produces vectors of roughly the same length as the main construction does in its worse case scenario, it does so with less computations during preprocessing. This makes it more attractive than the main construction for the specific case of binary HVE schemes.

One of the key insights that allowed the creation of both the main and the alternative construction is that any non-zero inner product matching algorithm can be transformed to a zero inner product matching algorithm if the non-zero result can be predicted at the time of token generation.

In the main construction this is used to turn an equation in Sedghi's basic intuition into a zero inner product matching algorithm. This is done by translating the left-hand side of the equation into an inner product and using the right hand side of the equation as the predictable result.

In the alternative construction, the zero inner product of the KSW Approach is first reduced to a non-zero inner product of half its length and then given a predictable result by fixing its randomization values.

In addition to the successful attempt to find a solution, there was one failed attempt. This attempt was based on extending the symmetric IPE scheme of Shi, Sahai and Waters in such a way that it would allow for non-zero inner products. This attempt first failed during the proving stage, as it could not reproduce the SSW security proof. It was then further debunked and proven to be broken using Katz, Sahai and Waters' master theorem of hardness.

Interestingly enough, the attack on the first attempt can be generalized to find potential attacks against other schemes in a bilinear setting. These attacks allow an adversary to detect a specific linear relation between plaintext query values and do not require the generation of ciphertexts, which means they can be used to detect not just specific queries, but also ranges of queries that only agree on that specific relation.

5.1 Future Research

As the constructions in this thesis are focused on preprocessing of inputs to IPE schemes, their contribution is largely as a tool to creating more efficient, secure HVE constructions using IPE. The existence of these constructions shows the possibility of constructing efficient HVE schemes within any IPE compatible construction, allowing future researchers to focus on new advances on either IPE itself or on entirely new but IPE compatible constructions.

One thing to keep in mind is that the efficiency of the main construction largely relies on pairing operations being considerably more expensive than regular modular multiplication and addition, making the cost of calculating of the various precomputations acceptable. As research into more efficient bilinear mapping functions advances, this premise may need to be revised and the alternative construction may become overall more efficient.

Bibliography

[ABC^{+}08] Michel Abdalla, Mihir Bellare, Dario Catalano, Eike Kiltz, Tadayoshi Kohno, Tanja Lange, John Malone-Lee, Gregory Neven, Pascal Paillier, and Haixia Shi. Searchable encryption revisited: Consistency properties, relation to anonymous ibe, and extensions. *Journal of Cryptology*, 21:350–391, 2008. 10.1007/s00145-007-9006-6.

[BB04] Dan Boneh and Xavier Boyen. Efficient selective-id secure identity-based encryption without random oracles. In Christian Cachin and Jan Camenisch, editors, *Advances in Cryptology - EUROCRYPT 2004*, volume 3027 of *Lecture Notes in Computer Science*, pages 223–238. Springer Berlin / Heidelberg, 2004.

[BIP09] Carlo Blundo, Vincenzo Iovino, and Giuseppe Persiano. Private-key hidden vector encryption with key confidentiality. In *Proceedings of the 8th International Conference on Cryptology and Network Security*, CANS '09, pages 259–277, Berlin, Heidelberg, 2009. Springer-Verlag.

[BIP10] Carlo Blundo, Vincenzo Iovino, and Giuseppe Persiano. Predicate encryption with partial public keys. Cryptology ePrint Archive, Report 2010/476, 2010.

[BSW10] Dan Boneh, Amit Sahai, and Brent Waters. Functional encryption: Definitions and challenges. Cryptology ePrint Archive, Report 2010/543, 2010.

[BW07] Dan Boneh and Brent Waters. Conjunctive, subset, and range queries on encrypted data. In Salil Vadhan, editor, *Theory of Cryptography*, volume 4392 of *Lecture Notes in Computer Science*, pages 535–554. Springer Berlin / Heidelberg, 2007.

[CIP11] Angelo De Caro, Vincenzo Iovino, and Giuseppe Persiano. Hidden vector encryption fully secure against unrestricted queries. 2011.

[IP08] Vincenzo Iovino and Giuseppe Persiano. Hidden-vector encryption with groups of prime order. In Steven Galbraith and Kenneth Paterson, editors, *Pairing-Based Cryptography Pairing 2008*, volume 5209 of *Lecture Notes in Computer Science*, pages 75–88. Springer Berlin / Heidelberg, 2008.

[KSW08] Jonathan Katz, Amit Sahai, and Brent Waters. Predicate encryption supporting disjunctions, polynomial equations, and inner products. In *Proceedings of the theory and applications of cryptographic techniques*

27th annual international conference on Advances in cryptology, EUROCRYPT'08, pages 146–162, Berlin, Heidelberg, 2008. Springer-Verlag.

[LL11] Kwangsu Lee and Dong Lee. Improved hidden vector encryption with short ciphertexts and tokens. *Designs, Codes and Cryptography*, 58:297–319, 2011. 10.1007/s10623-010-9412-x.

[LOS$^+$10] Allison Lewko, Tatsuaki Okamoto, Amit Sahai, Katsuyuki Takashima, and Brent Waters. Fully secure functional encryption: Attribute-based encryption and (hierarchical) inner product encryption, 2010. this is a full version of a paper appearing in Eurocrypt 2010. This version has appendices and other extra material that does not appear in the Eurocrypt version. alewko@cs.utexas.edu 14697 received 1 Mar 2010, last revised 29 Mar 2010.

[O'N10] Adam O'Neill. Definitional issues in functional encryption. Cryptology ePrint Archive, Report 2010/556, 2010.

[SBGN10] Dongdong Sun, Colin Boyd, and Juan Manuel Gonzalez Nieto. Predicate encryption for multi inner products. In *IFIP Advances in Information and Communication Technology : Proceedings of 25th IFIP TC-11 International Information Security Conference, SEC 2010*, volume 330, pages 229–240. Springer, 2010.

[SSW09] Emily Shen, Elaine Shi, and Brent Waters. Predicate privacy in encryption systems. In Omer Reingold, editor, *Theory of Cryptography*, volume 5444 of *Lecture Notes in Computer Science*, pages 457–473. Springer Berlin / Heidelberg, 2009.

[SvLN$^+$10] Saeed Sedghi, Peter van Liesdonk, Svetla Nikova, Pieter Hartel, and Willem Jonker. Searching keywords with wildcards on encrypted data. In Juan Garay and Roberto De Prisco, editors, *Security and Cryptography for Networks*, volume 6280 of *Lecture Notes in Computer Science*, pages 138–153. Springer Berlin / Heidelberg, 2010.