UNIVERSIDADE FEDERAL DE SANTA CATARINA
INFORMÁTICA E ESTATÍSTICA

Thaís Bardini Idalino

USING COMBINATORIAL GROUP TESTING TO SOLVE INTEGRITY ISSUES

Florianópolis

2015

Thaís Bardini Idalino

USING COMBINATORIAL GROUP TESTING TO SOLVE INTEGRITY ISSUES

dissertação submetida ao Programa de Pós-Graduação em Ciência da Computação para a obtenção do Grau de Mestre em Ciência da Computação.
Orientador: Prof. Ricardo Felipe Custódio, Dr.
Universidade Federal de Santa Catarina
Coorientadora: Prof[a]. Lucia Rosana Moura, Dr[a].
University of Ottawa, Canadá

Florianópolis

2015

Ficha de identificação da obra elaborada pelo autor,
através do Programa de Geração Automática da Biblioteca Universitária da UFSC.

Idalino, Thaís Bardini
Using combinatorial group testing to solve integrity issues / Thaís Bardini Idalino ; orientador, Ricardo Felipe Custódio ; coorientadora, Lucia Rosana Moura. - Florianópolis, SC, 2015.
85 p.

Dissertação (mestrado) - Universidade Federal de Santa Catarina, Centro Tecnológico. Programa de Pós-Graduação em Ciência da Computação.

Inclui referências

1. Ciência da Computação. 2. Garantia parcial de integridade. 3. Testes combinatórios de grupo. 4. Agregação de assinaturas. 5. Verificação de assinaturas em lotes. I. Custódio, Ricardo Felipe. II. Moura, Lucia Rosana. III. Universidade Federal de Santa Catarina. Programa de Pós-Graduação em Ciência da Computação. IV. Título.

Thaís Bardini Idalino

USING COMBINATORIAL GROUP TESTING TO SOLVE INTEGRITY ISSUES

Esta dissertação foi julgada aprovada para a obtenção do Título de "Mestre em Ciência da Computação", e aprovada em sua forma final pelo Programa de Pós-Graduação em Ciência da Computação.

Florianópolis, 1º de junho 2015.

Prof. Ronaldo dos Santos Mello, Dr.
Coordenador do Programa

Prof. Ricardo Felipe Custódio, Dr.
Universidade Federal de Santa Catarina
Orientador

Profª. Lucia Rosana Moura, Drª.
University of Ottawa, Canadá
Coorientadora

Banca Examinadora:

Prof. Julio César López Hernández, Dr.
Universidade Estadual de Campinas

Profª. Jerusa Marchi, Drª.
Universidade Federal de Santa Catarina

Profª. Carina Friedrich Dorneles, Drª.
Universidade Federal de Santa Catarina

Dedico este trabalho aos meus pais, familiares e amigos.

AGRADECIMENTOS

Gostaria de agradecer aqui a todos que, de uma forma ou de outra, foram fundamentais nessa etapa da minha vida.

Primeiramente, gostaria de agradecer o apoio fundamental dos meus pais, avós e demais familiares. Vocês me ensinaram a ser quem eu sou hoje e tudo o que conquistei até agora é reflexo dessa educação. Suas palavras de conforto e encorajamento foram indispensáveis para a finalização desse trabalho, muito obrigada por tudo.

Gostaria também de agradecer ao Felipe por estar ao meu lado sempre, nos momentos de alegria e de angústia e por me ajudar a vencer cada desafio encontrado no caminho. Seu companheirismo e paciência foram fundamentais no decorrer de mais essa etapa.

Um agradecimento especial também às minhas amigas, que sempre me ouviram nos momentos de desespero e apresentaram palavras de conforto e confiança. Obrigada por sempre acreditar e torcer por mim, nossas tardes de sábado sempre serão lembradas com muito carinho.

Não poderia esquecer de todos os amigos que fiz no LabSEC. Vocês contribuíram não só com esse trabalho, mas também com todas as outras etapas vencidas e conhecimentos obtidos nesses quase cinco anos de laboratório.

Por fim, agradeço ao meu Orientador Ricardo Custódio, Coorientadora Lucia Moura e colaborador Daniel Panario. Vocês me ajudaram e guiaram durante o desenvolvimento desse trabalho, além de proporcionar diversas outras oportunidades.

RESUMO

O uso de documentos eletrônicos para compartilhar informações é de fundamental importância, assim como a garantia de integridade e autenticidade dos mesmos. Para provar que alguém é dono ou concorda com o conteúdo de um documento em papel, essa pessoa precisa assiná-lo. Se o documento foi modificado após a assinatura, geralmente é possível localizar essas modificações através de rasuras. Existem técnicas similares em documentos digitais, conhecidas como assinaturas digitais, porém, propriedades como as de identificar as modificações são perdidas.

Ao determinar quais partes de um documento foram modificadas, o receptor da mensagem seria capaz de verificar se essas modificações ocorreram em partes importantes, irrelevantes ou até esperadas do documento. Em algumas aplicações, uma quantidade limitada de modificações são permitidas mas é necessário manter o controle do local em que elas ocorreram, como em formulários eletrônicos. Em outras aplicações modificações não são permitidas, mas é importante poder acessar partes das informações que tem integridade garantida ou até mesmo utilizar a localização das modificações para investigação.

Neste trabalho é considerado o problema de garantia parcial de integridade e autenticidade de dados assinados. Dois cenários são estudados: o primeiro está relacionado com a localização de modificações em um documento assinado e o segundo está relacionado com a localização de assinaturas inválidas em um conjunto de dados assinados individualmente.

No primeiro cenário é proposto um esquema de assinatura digital capaz de detectar e localizar modificações num documento. O documento a ser assinado é primeiramente dividido em n blocos, tendo em conta um limite d para a quantidade máxima de blocos modificados que o esquema de assinatura consegue localizar. São propostos algoritmos eficientes para as etapas de assinatura e verificação, resultando em uma assinatura de tamanho razoavelmente compacto. Por exemplo, para d fixo, são adicionados $\mathrm{O}(\log n)$ *hashes* ao tamanho de uma assinatura tradicional, ao mesmo tempo permitindo a identificação de até d blocos modificados.

No cenário de localização de assinaturas inválidas em um conjunto de dados assinados individualmente é introduzido o conceito de níveis de agregação de assinatura. Com esse método o verificador pode distin-

guir os dados válidos dos inválidos, em contraste com a agregação de assinaturas tradicional, na qual até mesmo um único dado modificado invalidaria todo o conjunto de dados. Além disso, o número de assinaturas transmitidas é muito menor que num método de verificação em lotes, que requer o envio de todas as assinaturas individualmente. Nesse cenário é estudada uma aplicação em bancos de dados terceirizados, onde cada tupla armazenada é individualmente assinada. Como resultado de uma consulta ao banco de dados, são retornadas n tuplas e um conjunto de t assinaturas agregadas pelo servidor (com t muito menor que n). Quem realizou a consulta executa até t verificações de assinatura de maneira a verificar a integridade das n tuplas. Mesmo que algumas dessas tuplas sejam inválidas, pode-se identificar exatamente quais são as tuplas válidas. São propostos algoritmos eficientes para agregar, verificar as assinaturas e identificar as tuplas modificadas.
Os dois esquemas propostos são baseados em testes combinatórios de grupo e matrizes *cover-free*. Nesse contexto são apresentadas construções detalhadas de matrizes *cover-free* presentes na literatura e a aplicação das mesmas nos esquemas propostos. Finalmente, são apresentadas análises de complexidade e resultados experimentais desses esquemas, comprovando a sua eficiência.

Palavras-chave: Assinaturas digitais. Garantia parcial de integridade. Localização de modificações. Testes combinatórios de grupo. Famílias cover-free. Agregação de Assinaturas. Verificação em lotes. Bancos de dados terceirizados.

ABSTRACT

We consider the problem of partially ensuring the integrity and authenticity of signed data. Two scenarios are considered: the first is related to locating modifications in a signed document, and the second is related to locating invalid signatures in a set of individually signed data.
In the first scenario we propose a digital signature scheme capable of locating modifications in a document. We divide the document to be signed into n blocks and assume a threshold d for the maximum amount of modified blocks that the signature scheme can locate. We propose efficient algorithms for signature and verification steps which provide a reasonably compact signature size. For instance, for fixed d we increase the size of a traditional signature by adding a factor of $O(\log n)$hashes, while providing the identification of up to d modified blocks.
In the scenario of locating invalid signatures in a set of individually signed data we introduce the concept of levels of signature aggregation. With this method the verifier can distinguish the valid data from the invalid ones, in contrast to traditional aggregation, where even a single invalid piece of data would invalidate the whole set. Moreover, the number of signatures transmitted is much smaller than in a batch verification method, which requires sending all the signatures individually. We consider an application in outsourced databases in which every tuple stored is individually signed. As a result from a query in the database, we return n tuples and a set of t signatures aggregated by the database server (with t much smaller than n). The querier performs t signature verifications in order to verify the integrity of all n tuples. Even if some of the tuples were modified, we can identify exactly which ones are valid. We provide efficient algorithms to aggregate, verify and identify the modified tuples.
Both schemes are based on nonadaptive combinatorial group testing and cover-free matrices.

Keywords: Digital signatures. Partial data integrity. Modification localization. Combinatorial group testing. Cover-free families. Aggregation of signatures. Batch verification. Outsourced databases.

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

SUMÁRIO

1 INTRODUCTION

The use of electronic media for sharing information is a key factor in communication and service provision. In particular, the Internet provides a large dissemination of digital information, as well as the storage of data in outsourced servers. However, besides all the convenience and facility we can achieve, it has also increased our concern about assuring the security of the shared and stored information since sometimes we cannot have control on who can access, copy or modify all these data.

These data are often represented in the form of electronic documents, which are a set of binary digits generated by a computer or any media or device for any electronic processing. Electronic documents follow no format or readability requirements except when retrieved for human-use. Just as paper documents, these electronic documents can be digitally signed. So, you can verify some security attributes of these documents, such as authorship, agreement with the content and integrity. To facilitate understanding, it is useful to draw an analogy of these security properties between paper documents and electronic documents. For instance, to show that a person is the owner or agree with the content of a document in paper, she just has to sign it in the traditional way. If this document was modified after the signature process, then it is usually possible to locate the modifications through erasures on paper. In electronic documents there are similar procedures, known as digital signatures, where with a few calculations it is possible to sign a document and prove its origin and integrity. However, the property of locating which parts of the document have been modified is lost. Digital signature schemes will detect if modifications were done in a signed document, but do not offer information on where exactly those modifications occurred. In this context, even a single bit change would invalidate the whole document.

The guarantee of security in documents and messages sent through a data communication network is essential, but it can be even more delicate when storing data in outsourced servers, where the server itself is not reliable. In this context, instead of single documents, we sometimes have a large set of independently signed documents (as we can see in Step 1 of Figure 1). Consequently, we need mechanisms to decrease the amount of signatures sent through network (Step 2 of Figure 1) and speed up the verification process (Step 3 of Figure 1). In the literature, we can find methods of *aggregation of signatures* and

batch verification that aim to improve the verification of several signatures in a more efficient way. The former provides a single signature by aggregating all the signatures into one before sending to the verifier, decreasing the number of signatures transmitted. However, if the verification fails there is not enough information to identify the faulty ones (ZAVERUCHA; STINSON, 2009). With the latter one, we send the signatures individually and aggregate them only during the verification, allowing the identification of faulty signatures if necessary but generating a considerable communication overhead.

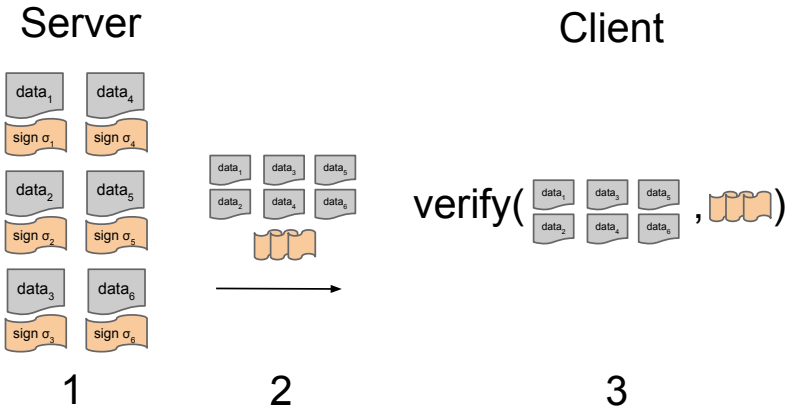

Figure 1: Relation between client and server with several signed data.

In this work we propose two new signature schemes in order to solve the problems above. The first scheme is called Modification Location Signature Scheme (MLSS) and aims to guarantee partial integrity of data. It considers a document divided into blocks and employs combinatorial group testing to determine if they were modified, and if they have been, to identify which of them have been modified. We propose efficient algorithms for signing and verification, resulting in a compact signature size, and present experimental results to show its efficiency.

The second scheme introduces the concept of levels of signature aggregation applied to scenarios where the verification of a large quantity of independently signed data is needed. We consider this method in a real scenario of outsourced databases, where the tuples are independently signed. With this method, the querier can analyze the query results and distinguish the valid tuples from the invalid ones, in

contrast to traditional aggregation, where even a single invalid tuple would invalidate the whole query. The method provides a *trade-off* between signature aggregation and batch verification, where less signatures are sent through the network without losing the ability to identify the faulty signatures. Both presented schemes are based on nonadaptive combinatorial group testing and cover-free families.

1.1 OBJECTIVES

1.1.1 General Objectives

The general objective of this work is to propose signature schemes that provide more information on which part of the data has been modified, on one hand, finding the location where changes occurred, and on the other hand, guaranteeing which parts of the data are integral. The first specific contribution of this work is a signature scheme able to allow changes in electronic documents without invalidating the signature of the document as a whole. Moreover, it is possible to detect which parts have been changed and which were not. The second specific contribution of this work is a solution for a *trade-off* between the techniques of aggregation of signatures and batch verification, applied in scenarios where a large amount of signature verifications are needed. In both problems we use nonadaptive combinatorial group testing algorithms to generate and verify the digital signatures.

1.1.2 Specific Objectives

1. Propose a new method to identify modifications in signed documents with the following objectives:

    - To propose a new signature algorithm with artifacts (the signature itself) that can be used to verify the individual integrity of different parts of the document;
    - to identify the modified blocks of a document in an efficient way;
    - to evaluate the efficiency of the method;
    - to propose methods to divide the document into blocks, considering different applications.

2. Propose a method to verify and locate invalid signatures in a large amount of individually signed data, decreasing the bandwidth overhead and the verification time:

    - Propose algorithms to aggregate different signatures without losing the ability to identify the invalid ones;
    - compare the proposed algorithms with the traditional techniques of aggregation of signatures and batch verification;
    - consider a real scenario where the method can be applied;
    - evaluate the efficiency of the method.

1.2 LIMITATIONS

In this work we focus on guaranteeing integrity of data, and since we use asymmetric key algorithms, we also provide authenticity and non-repudiation. This guarantees are achieved by verifying the signature as described in Chapters 4 and 5. We do not consider any other security guarantee, such as privacy, authorization and completeness, but this could be achieved by adding the desired algorithm in specific parts of our methods.

1.3 JUSTIFICATION

It is desirable that the receiver of a signed document or a set of signed data be able to check their signature with the least effort. In addition, if part of the data does not match its signature, it is also desirable to determine which parts of the data have been modified and which parts are integral.

Nowadays, digital signature algorithms allows the verification of integrity, authenticity and non-repudiation in signed documents. However, they are not designed to allow the location of modified portions of a document. Additionally, in order to verify several digital signatures simultaneously there are methods of *aggregation of signatures* and *batch verification*. With the former it is possible to send less data through the network, but only with batch verification it is possible to identify the invalid signatures in order to not invalidate the entire set of messages/signatures.

This work is motivated by the hypothesis that it is possible to localize modifications in a signed document, increasing the standard

signature size by a small factor and not increasing the cost of verifying the signature of a document in terms of cryptographic processing. We show that it is possible to construct the signature, verify and localize the modifications with combinatorial group testing algorithms. We also demonstrate that it is possible to verify several aggregated signatures simultaneously without losing the ability of identifying the invalid ones. In this way, it is possible to solve the trade-off between aggregation of signatures and batch verification in order to enjoy the benefits of both methods.

In order to approach these problems, it is necessary to carefully design methods to locate modifications while sending a small amount of information through network and keeping the signature verification process efficient.

1.4 METHODOLOGY

In order to achieve the goals of this work, we first studied papers, books and theses related to digital signatures, aggregation of signatures, batch verification, and identification of invalid signatures in a batch. We also studied possible applications, such as database integrity, big data, cloud computing and forensics. Additionally, we studied group testing algorithms, in particular the generation of cover-free matrices and their use to verify signatures. Then, we selected the most relevant works in order to identify the problems and build our solutions.

By studying the problems and related works, we built a scheme capable of locating modifications on signed documents. We presented algorithms to generate and verify the signatures, their proofs of correctness and their complexity. The complexity analysis provides a general comparison in terms of hash computations and cryptographic operations. In order to have a more tangible time comparison, we give some experimental results. We quantify the running time that the algorithm would take based on openssl using the RSA signature and verification algorithms (Chapter 4).

In the context of identifying invalid signatures in a set of signed data, we introduce the concept of level of aggregation building on the existing ideas of aggregation of signatures and batch verification. We identify that this technique would be suitable to an application on outsourced databases. We partially aggregate and verify the signatures, proving its efficiency by comparing the number of operations of the proposed algorithms with the ones found in the literature for this

specific application (Chapter 5).

1.5 SCIENTIFIC CONTRIBUTIONS

The Modification Location Signature Scheme proposed in Chapter 4 presents a scheme to efficiently localize modifications in signed documents. It was published as an article in *Information Processing Letters* (IPL):

Idalino, T. B., Moura, L., Custodio, R.F., Panario, D., "Locating modifications in signed data for partial data integrity". *Information Processing Letters*, Published online in march 2015. Available in: `http://dx.doi.org/10.1016/j.ipl.2015.02.014`.

This work was also presented as a student talk at the Third *International Conference on Cryptology and Information Security in Latin America*, held in Florianópolis, Brazil from September 17 to September 19.

The Level-d signature aggregation scheme presented in Chapter 5 provides a scheme to partially aggregate digital signatures without losing the ability to identify the invalid ones. In this context, an application in an outsourced database scenario is considered. This scheme is part of a paper in preparation for publication.

1.6 OUTLINE

This work is organized as follows. In Chapter 2, we present basic concepts related to information security and prior works related to aggregation of signatures, batch verification and other related works. In Chapter 3, we introduce combinatorial group test and three cover-free matrix constructions, with detailed information about the use of each one. In Chapter 4, we address the problem of locating modifications in signed documents, presenting our first main contribution, experimental results and comparison between our method and traditional schemes. In Chapter 5, we present our second main contribution, the level-d aggregation scheme. We also present comparisons between previous methods and the proposed one and show the solution of the trade-off between aggregation and batch verification. Finally, Chapter 6 includes the final considerations and suggestions for future work. We want to reinforce that, since two different methods for different applicati-

ons were proposed, their analysis are individually presented in their respective chapters.

2 BASIC CONCEPTS IN SECURITY AND RELATED WORK

2.1 INFORMATION SECURITY CONCEPTS

Information security is an old concern that has been improved over the years. Its main goals are related to the guarantee of integrity, authenticity and non-repudiation (with a few naming variations in the literature). Assuring data *integrity* means that the receiver of a message needs mechanisms to verify that it was not modified by an unauthorized party, such as an intruder; *authenticity* is related to the guarantee that the information received came from the expected origin; and *non-repudiation* assures that the sender of a message can not deny later that he/she sent the message (MARTIN, 2012; SCHNEIER, 1996; MENEZES; VANSTONE; OORSCHOT, 1996). In this work our main concern is the integrity of data, so we will present this concept in more details next.

Data integrity consists in ensuring accuracy and consistency of data. According to Schneier (SCHNEIER, 1996), the person who receives a message needs to be able to verify if it was not modified. Martin (MARTIN, 2012) presents four different levels of data integrity. The first level protects against *accidental errors*, which can occur from noise in the communication channel. Examples of mechanisms to ensure the data integrity are error-correcting codes and the use of check sums. The second level of data integrity protects against *simple manipulations*, which are related to accidental errors when the data are changed in a predicted way. Hash functions are mechanisms used to verify the integrity in this case. *Active attacks* are in the third level, where the mechanisms must prevent an attacker from creating a valid integrity digest from a message manipulated by them. To solve this problem, usually it is required an authentication of the origin of the data. The most widely used cryptographic mechanism against these types of errors is the Message Authentication Code (MAC) (WEGMAN; CARTER, 1981). The fourth level of data integrity protects against *repudiation attacks* that occurs when the creator denies the responsibility on creating the integrity digest of the data. In this case we use digital signatures, which provide integrity and can be verified by third parties.

The mechanism most commonly used to ensure the integrity of data is the *hash* function (or digest). This function maps texts of arbitrary size into fixed-size text and in theory it should be easy to compute but hard to invert (PRENEEL, 2010). Hash functions are cryptographic

primitives that have many important and varied uses, such as to provide data integrity, generate pseudorandom numbers, construct digital signatures, and others.

Martin (MARTIN, 2012) presents some practical and security properties that a hash function should have. The first practical property consists in, regardless of the amount of data that is provided as an input for the hash function, it must return as a result a fixed size data. Usually, the result of the function (known as "hash") is much smaller than the entry data. The second practical property requires that a hash function should be very easy to compute (computed in polynomial time).

The security properties presented by Martin (MARTIN, 2012) are pre-image, second pre-image and collision resistance. These properties are illustrated in Figure 2 (adapted from Martin (MARTIN, 2012), page 191). The first entails that it should be hard (in computational efficiency) to invert a hash function. For example, given the output z, it should be hard to find an input x where $hash(x) = z$. The second pre-image resistance is similar to the previous one. Here, given an input x and its hash z, it should be hard to find another input x' that has the same hash z. The collision resistance states that it should be hard to find two different inputs x e x' that result in the same hash z. The difference between the second pre-image and the collision resistance is that in the first one the input and its hash are known, and on the second one just the hash is known.

A hash function by itself cannot guarantee data integrity, since an attacker can simply modify the original message and recalculate the hash. To solve this problem we can use *Message Authentication Codes* (MACs) (WEGMAN; CARTER, 1981). It consists in two algorithms (SIGN and VERIFY) and a secret key k previously chosen by sender and receiver. It relies on symmetric encryption, where the same key k is used to encrypt and decrypt the messages. Given a message m, the signer computes $\sigma = \text{SIGN}(k, m)$ and produces a *tag* σ. The receiver accepts the pair (m, σ) as valid if $\text{VERIFY}(k, m, \sigma)$ is true. If m was modified during the communication, σ can not be recreated by a third party, since only the sender and receiver know k.

A MAC can not protect against repudiation attacks due to the fact that senders and receivers share the same secret key k, not allowing to identify which of them created the *tag*. In order to identify exactly which part created the integrity digest of a message, a mechanism similar to a handwritten signature is necessary, known as *digital signature*. With digital signatures it is possible to guarantee the integrity,

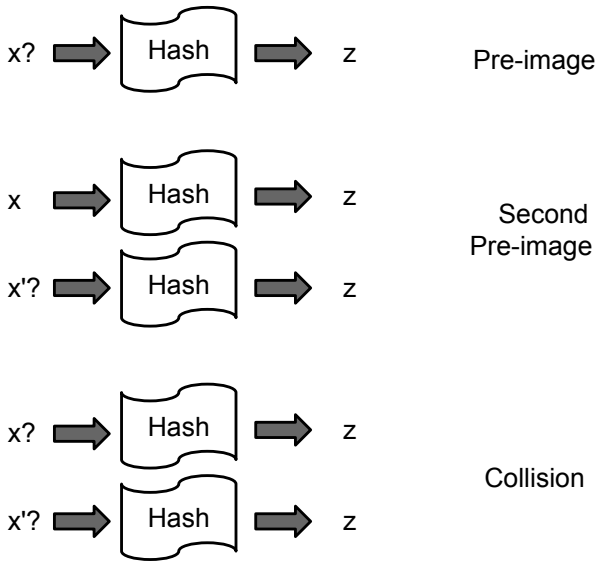

Figure 2: Security properties of a hash function.

authenticity and non-repudiation (MARTIN, 2012). This mechanism is presented in detail in the next section.

2.2 DIGITAL SIGNATURE

The first notion of a digital signature scheme emerged in 1976, with the key distribution algorithm of Diffie and Hellman (DIFFIE; HELLMAN, 1976). A famous practical implementation of a digital signature scheme was presented by Rivest, Shamir and Adleman in 1978 (RIVEST; SHAMIR; ADLEMAN, 1978), known as RSA algorithm, enabling the sending of signed messages in order to guarantee the authenticity, integrity and non repudiation of the data after signed. It uses asymmetric encryption, also known as public-key encryption, which requires a pair of keys: one public and one private. To send a signed electronic message the sender generates a public and a private key and generates the signature by encrypting the hash of the message with his private key. Anyone can use the corresponding public key to verify the signature, but since only the signer has the private key, only him could have

generated the signature.

In more detail, the RSA algorithm (RIVEST; SHAMIR; ADLEMAN, 1978) works as follows. First, we need to choose two prime numbers p and q and calculate n as their product $n = pq$, the value n is public while p and q are secret. Prime numbers must be large enough such that it is difficult, from a computational point of view, to factor their product. Second, we choose an integer e such is relatively prime with $(p-1)(q-1)$ and determine d as the multiplicative inverse of e, such that $de \equiv 1 \pmod{(p-1)(q-1)}$. We can say that the public key is the pair (e, n) and the private key is (d, n). Considering m a message to be send through the network and m' the message received on the other side. The signature of m can be calculated by $c = hash(m)^d \mod n$ and the pair (m, c) is sent to the verifier. At the other end we receive (m', c), and in order to verify the signature we only need to verify if $hash(m') = c^e \mod n$.

There are several important digital signature methods, as the well know El-Gamal (ELGAMAL, 1985), DSA (KERRY; GALLAGHER, 2013) and ECDSA (JOHNSON; MENEZES, 1999), or methods based on *trapdoor* permutation (BELLARE; ROGAWAY, 1993) and bilinear maps (BONEH; LYNN; SHACHAM, 2001), which we will not see in details here.

Digital signatures have been used in several applications that need to verify a large quantity of signatures in a more efficient way than verifying each individual signature. A few examples are outsourced databases, where each entry (or a subset) is signed; a sensor network where each sensor need to send several signed messages; vehicular communication, among others. To satisfy this requirement, there are techniques to aggregate signatures and execute batch verification, which we detail in next sections.

2.3 SIGNATURE AGGREGATION

Various methods in the literature propose the aggregation of digital signature as a solution to send less data through the network and improve the verification steps (BONEH et al., 2003; MYKLETUN; NARASIMHA; TSUDIK, 2006). Instead of sending n signatures, the methods combine these signatures into one without losing the ability to verify the integrity of the corresponding data. On the verification side, instead of verifying n signatures individually, only one verification is needed. It is only possible to aggregate the signatures as long as it has the so-called homomorphism property.

Homomorphic encryption is a technique proposed by Rivest et al. (RIVEST; ADLEMAN; DERTOUZOS, 1978). It is an encryption method that allows calculations to be performed on encrypted data, which creates an encrypted result that, when decrypted, corresponds to the results of operations as if they had been made directly in plaintext. Consider some operator $\odot$, messages m_1 and m_2, and an encryption function $e(\cdot)$. We say that an encryption scheme has the homomorphic property if the following is true: $e(m_1 \odot m_2) \equiv e(m_1) \odot e(m_2)$. In other words, we obtain the same result if we first apply the operation on the messages and then encrypt the result, or if we first encrypt the messages separately and then apply the operation.

The concept of *aggregation of signatures* emerged with Boneh et al. (BONEH et al., 2003). They show that given n signatures generated individually, from n distinct messages and users, it is possible to aggregate all these signatures in a single signature. This decreases the amount of signature data to be transmitted over the network to the recipient of the documents. In addition, the recipient can verify the signature of n documents in a single signature verification operation. The algorithm proposed by Boneh et al. (BONEH et al., 2003) is based on the *Co-Gap Diffie-Hellman* digital signature algorithm, using bilinear maps to allow the signatures aggregation (see (BONEH et al., 2003) for more details). The aggregation can be executed by a third party who knows all the signatures, public keys and messages, and can convince the verifier that each signer signed their corresponding message. As a result, we compress the n signatures into one with the size of a traditional signature. Its verification is given in linear time (n signatures), but if all of them are signed by the same key, it is possible to further improve the verification process. One issue that may be a challenge with this method is that the aggregation can be done by anyone, not necessarily by one of the signers of the messages, and not even a person of confidence of them.

Lysyanskaya et al. (LYSYANSKAYA et al., 2004) present a sequential aggregate signature scheme in which the set of signers is arranged in a specific order. Thus, the aggregation of signatures is made in a sequential and incremental way, becoming ideal to use in certificate chains. The algorithms to generate keys, sign and verify the signatures are based on *full-domain signatures*, in which are applied to trapdoor permutation families. These families present some permutation properties that make it possible to sign, verify and aggregate the signatures. The authors present methods to generalize the scheme and use it with RSA, treating it as a permutation family. However, there are some

limitations, as an example we need to restrict the values of the keys and replace operations that do not apply to RSA (see (LYSYANSKAYA et al., 2004) for more details). The size of the final aggregated signature is equal to a traditional signature.

Lu et al. (LU et al., 2012) show a scheme to sequentially aggregate signatures that is provably secure, in which the verifier does not need to know the order of the aggregated signatures. They use bilinear maps to construct the signatures and its aggregation, where the signatures are smaller than in Lysyanskaya et al (LYSYANSKAYA et al., 2004) and the verification process is faster than in Boneh et al. scheme (BONEH et al., 2003).

Neven (NEVEN, 2008), on the other hand, presents a scheme where the goal is to decrease not only the signature size, but also the size of all the data that are sent through the network. He observes that the methods previously presented do not offer freedom on choosing the cryptographic parameters. Boneh et al. (BONEH et al., 2003) and Lu et al. (LU et al., 2012), as an example, require that all the signers use the same elliptic-curve groups. Lysyanskaya et al. (LYSYANSKAYA et al., 2004) require the signers to be arranged in such a way that their keys are in ascending order, which is the opposite of what we need in a real certificate chain (since the first key, from the Certificate Authority, is usually bigger than the final users keys). Neven also uses trapdoor permutations and, comparing to previous work, presents a better efficiency to generate and verify the signatures, in addition to a bandwidth overhead of only 160 bits.

Katz and Lindell (KATZ; LINDELL, 2008) propose an aggregation scheme using MACs for authenticated communication on mobile ad-hoc networks. In this scenario, there is a root node responsible for communication with the base station, and internal nodes that want to communicate. The communication will involve a big quantity of bits since each node forward the data and MACs of its children upwards until the root is reached. To solve this drawback, they aggregate all the MACs of the messages sent to all the nodes. Therefore, the central station can, through one final MAC, verify all the messages from all the nodes, decreasing the amount of data passed from node to node, by recalculating the MACs, concatenating and verifying them only once. There is no improvement in the verification process, but the amount of data sent from node to node decreases significantly.

Yavuz and Ning (YAVUZ; NING, 2009) propose a computationally efficient signature scheme with public verifiability. This scheme is implemented in a network of wireless sensors used, for example, in military

applications. In this environment, it is not always possible the immediate communication of the sensor with its receptor and sometimes, it can takes a long period of time. Then, these sensors accumulate data and, if compromised, their keys can be used to forge messages. Therefore, it is necessary to guarantee the integrity of information stored before the compromising. In order to achieve this, Yavuz and Ning (YAVUZ; NING, 2009) propose schemes based on forward security and aggregation of signatures. Forward security guarantees that the data were stored before the compromising, and they use aggregation to aggregate all the signatures generated by the sensors in order to improve the verification. They use MACs as cryptographic primitives, ensuring computational efficiency.

In the context of outsourced databases, Mykletun et al. (MYKLETUN; NARASIMHA; TSUDIK, 2006) propose the *condensed-RSA* signature scheme in order to improve the bandwidth and verification of the signed database entries. In this scheme, each database entry m_i is signed by applying a cryptographic hash function, and then using the owner's secret key (s_k) with standard RSA encryption: $\sigma_i = hash(m_i)^{sk} \pmod{l}$. When a query is executed, the server returns n entries and only one aggregated signature generated as follows (this is possible only because of the RSA multiplicatively homomorphic property): $\sigma \equiv \prod_{i=1}^{n} \sigma_i \pmod{l}$. On the querier side, the verification is also improved by the aggregation, since only one decryption (using public key p_k) is required for the verification: $\sigma^{p_k} \equiv \prod_{i=1}^{n} hash(m_i) \pmod{l}$, in place of n standard RSA decryptions needed in the verifications: $\sigma_i^{p_k} \equiv hash(m_i) \pmod{l}$, $1 \leq i \leq n$.

2.4 BATCH VERIFICATION

Batch verification is another method that aims to reduce the computational complexity in systems that require many simultaneous signature verifications. It was introduced by Fiat (FIAT, 1989) using RSA signatures, where it achieves its goal by reducing the total number of exponentiations needed during the verification process. The difference between this method and the previous one is that the aggregation of the signatures in a batch verification is performed by the verifier and all the signatures are individually sent to him.

Another batch verification method is presented by Harn (HARN, 1998a), that aims to verify several DSA signatures quickly. In the tra-

ditional way, each DSA verification requires two modular exponentiations, so, to verify n signatures we need $2n$ exponentiations. The method presented by Harn requires only 2 operations to verify n signatures, but if the verification fails it is necessary to verify each signature individually to locate the invalid ones within the batch. Harn also presents a batch verification method based on RSA signatures (HARN, 1998b), in which a set of signatures from one signer are multiplied to improve the verification process. Since all the signatures are signed by the same key, it is possible to multiply them due to the RSA multiplicatively homomorphic property: $(\prod_{i=1}^{n} \sigma_i)^e \equiv \prod_{i=1}^{n} \text{hash}(m_i) \pmod{l}$.

Bellare et al. (BELLARE; GARAY; RABIN, 1998) present three probabilistic methods to speed up the calculus of several modular exponentiations in cyclic groups during the verification of signatures issued by different signers. These methods are called *random subset test*, *small exponents test* and *bucket test*. They also present an alternative to RSA signature verifications called *screening*. In this case, given a set of messages and respective signatures to be verified, with only one exponentiation (instead of n), it is possible to verify all the signatures. This is a really efficient alternative, but if the signer has already signed in the past any message $m_i \in M$, the test may accept an actual signature σ_i even if it is not valid anymore for the message m_i. In other words, the method can identify only if at some moment the signer already signed the messages in M. According to Zaverucha and Stinson (ZAVERUCHA; STINSON, 2009), several batch verification schemes today are based in methods presented by Bellare et al. (BELLARE; GARAY; RABIN, 1998).

Camenisch et al. (CAMENISCH; HOHENBERGER; PEDERSEN, 2011) present a method based in Bellare et al. (BELLARE; GARAY; RABIN, 1998) in which, besides speeding up the verification process, the signatures are generated with a reduced size, decreasing the amount of data sent through the network. Ferrara et al. (FERRARA et al., 2009) show how to batch verify different types of signature (as *identity-based* and *ring signatures*). They compare algorithms of batch verification and individual verification, showing that the former is really more efficient than the latter, even though performing extra operations (as multiplications) in order to execute a batch verification.

2.5 THE CASE OF INVALID SIGNATURES

According to Malina (MALINA; HAJNY; ZEMAN, 2013), invalid signatures cause a loss of efficiency in the verification process because they invalidate the entire set of signatures. Generally, to verify a batch, condensed or aggregated signature, the verifier has to multiply the messages' hashes and compare with the signature(s) received. If the comparison does not match, it means that there exists at least one invalid signature among the set of signatures. In order to not invalidate all the set of signatures it is necessary to precisely identify the faulty ones. This could only be done if the verifier of the signature has access to all the signatures individually, which is possible with batch verification but not with aggregated signatures (MYKLETUN; NARASIMHA; TSUDIK, 2006; ZAVERUCHA; STINSON, 2009). We can observe a *trade-off* between the two presented methods: by using aggregation we decrease the amount of data sent through the network, but only with batch verification we can identify all the faulty signatures.

Considering the case when we have all the signatures to compare, the easiest way is to verify each signature individually, which is obviously not efficient if there are hundreds of messages (and consequently signatures). Several studies have proposed methods using divide-and-conquer, identification codes, binary search, exponent testing among others. Divide-and-conquer methods were first introduced by Pastuszak et. al (PASTUSZAK et al., 2000) and improved by Law and Matt (LAW; MATT, 2007). Here, a set of signatures is recursively divided in half, until the invalid signatures are found. This method was implemented and evaluated by Ferrara et al. (FERRARA et al., 2009) and improved by Matt (MATT, 2009) in the cases where the number of invalid signatures is large. Zaverucha and Stinson (ZAVERUCHA; STINSON, 2009) are the first to observe that the problem of finding invalid signatures in a batch is a special case of a group testing problem, associating previous methods with existing group testing algorithms.

In this work, taking into consideration the *trade-off* between aggregation and batch verification and the importance of both, we propose new methods where a reasonable amount of data is transmitted while allowing the verification and identification of invalid signatures and document portions. The proposed methods are all based on nonadaptive combinatorial group testing.

2.6 OTHER RELATED WORKS

A few researchers have shown interest in locating modifications in different types of documents. Barreto (BARRETO, 2003), for example, use topological watermarks to detect and locate modifications in images. He also presents some possible attacks in the existent watermark methods and propose a hash chaining scheme that is resistant to these attacks.

Lytle et al. (LYTLE et al., 2012) show an example of the importance in locating modifications in some specific parts of a document. They present the case where some applications, as word processors or spreadsheets, allow data operations with customized codes (such as macros). These codes can be considered a security risk, since malicious codes can be introduced into the document and executed when the document is opened. They propose to sign the portion of document where the executable code is and verify the signature before executing it.

Goodrich et al. (GOODRICH; ATALLAH; TAMASSIA, 2005) propose a nonadative combinatorial group testing construction and use it combined with message authentication codes (MACs) to build schemes for data forensics marking. In this scheme, they combine the data using the group testing structure and calculate their authentication values using MACs (see more details about group testing in Chapter 3). In order to avoid the storage of the authentication values, they encode these bits in the data structure itself, proposing solutions for binary trees, lists, arrays and hash tables. Later this authentication values can be used to identify, among the n pieces of data from the data structure, the modified ones and this information can be used in forensic investigations.

Crescenzo (CRESCENZO; JIANG; SAFAVI-NAINI, 2009) proposes the notion of corruption-localizing hashing applied to software reliability and virus detection. It is useful in the first application for the cases where the download of a software fails, since by localizing the corrupted or missing blocks we know exactly what needs to be retransmitted; in the second application it can be used to detect undesired changes in files due to virus infection. Bonis and Crescenzo later improved this idea (BONIS; CRESCENZO, 2011a, 2011b) by proposing localizing codes based on combinatorial group testing techniques, which present some similarities with our method presented in Chapter 4. However, we want to remark that our algorithms were proposed completely independently from theirs before we were aware of their work. Indeed our schemes are applied in distinct applications, and here we focus on

offering mechanisms to localize corruptions using the document digital signature.

3 COMBINATORIAL GROUP TESTING

The history of group testing began in the World War II, 1942. The origin is usually credited to Robert Dorfman, although there are some doubts about this. The original idea was created to verify blood samples, among millions of people, in order to identify a few cases of syphilis in an economic way (DU; HWANG, 2000). Usually, to verify if n people have the disease, it would be necessary to perform n blood tests. The proposed technique indicates that, after collecting all the individual blood samples, they are pooled in t groups of n' elements. The groups are tested by mixing the blood samples, instead of testing each sample individually. If none of the n' elements in group t_i have the disease, the test applied in this group will be "negative"(or "pass"). Otherwise, if one or more samples are infected with the disease, the test applied in their group will be "positive"(or "fail"). Then, the n' elements from the positive tested group t_i must be tested again, to determine which of them are infected.

The purpose of group testing is to identify d defective elements from a set of n elements pooled into t groups where $t < n$. The groups are tested, instead of all elements individually. There are two types of group testing algorithms: *adaptive* and *nonadaptive*. In *adaptive* group testing, the results of the previous tests are used to determine subsequent tests, so, the tests are executed sequentially. In *nonadaptive* group testing, all the tests are specified ahead of time, which allows them to be run in parallel (for more information see the book by Du and Hwang (DU; HWANG, 2000)). In our method we need to use nonadaptive group testing, as we will explain in later chapters.

Among the nonadaptive methods, there is the one based on Cover-Free family (CFF) matrix representation. This representation was first introduced in Kautz and Singleton (KAUTZ; SINGLETON, 1964) with the name of *nonrandom binary superimposed codes*, used in storage and communication systems. Years later, various definitions were created and generalized (STINSON; WEI, 2002), (ERDöS; FRANKL; FUREDI, 1985), and today Cover-Free Families can also be found with the terms *binary superimposed codes* (KAUTZ; SINGLETON, 1964) and *disjunct matrices* (DU; HWANG, 2000), and its definition is presented bellow.

Definition 1. *A* d-cover-free family*, denoted* d-$CFF(t, n)$ *is a* $t \times n$ *binary matrix* M *with* $n \geq d + 1$*, such that for any set of column*

indexes C with $|C| = d$ and any column $c \notin C$, the following property holds: there exists a row i satisfying $M_{i,c} = 1$ and $M_{i,j} = 0$ for all $j \in C$.

We form the tests according to the rows of matrix M, i.e. for each $1 \leq i \leq t$, test i consists of exactly the items j for which $M_{i,j} = 1$. As stated in the next proposition, the properties of cover-free families assure that if the number of defectives is at most d then it is possible to determine the non-defective items from the passing tests. Then, we can conclude that all the other items are defective.

Proposition 1. *((DU; HWANG, 2000), Section 7.1) Consider a d-CFF(t, n) matrix M and its associated t-tuple T of test results. Let $P = \{j \in \{1, \ldots, n\} : \exists i$ such that $T_i =$ "pass" and $M_{i,j} = 1\}$ and let $I = \{1, \ldots, n\} \setminus P$. If $|I| \leq d$ then I is the set of defective items; otherwise, there are more than d defectives and I contains the set of defective items (and possibly a few other items).*

Proof. The proof of this proposition follows directly from Definition 1. If there are at most d defective items, any non-defective item c must be in a test without any of the defective ones, and so it will be in the set P; all the other items that are not in P are consequently defective. If $|I| > d$ we can conclude there are more than d defective items and the only ones we can guarantee that are not defective are the ones in P. □

An example of the application of a CFF matrix is given in Figure 3. In this example, 10 elements are tested by verifying 5 tests combining the elements prescribed by the matrix for each of the tests. This matrix is 1-cover-free but is not 2-cover-free. Any single defective element can be identified from the results of the verification; for example if the invalid element is the 9th one, only tests 3 and 5 will fail; tests 1, 2, 4 will pass, which is sufficient to determine that the other elements are valid. On the other hand, if 2 elements are invalid, since the matrix is not 2-cover-free we are not always able to determine exactly the set of invalid elements. If for example, elements 3 and 4 are invalid, tests 1, 4, 5 fail; tests 2 and 3 show that elements 1, 2, 5, 6, 7, 8 and 9 are valid and the returned set is I = $\{3, 4, 10\}$. This set has cardinality larger than $d = 1$, so the verifier knows it contains all invalid elements; but may contain valid ones; indeed the valid element 10 is included here.

We give next a few useful explicit constructions of cover-free families found in the literature.

element index:	1	2	3	4	5	6	7	8	9	10
test 1:	1	1	1	1	0	0	0	0	0	0
test 2:	1	0	0	0	1	1	1	0	0	0
test 3:	0	1	0	0	1	0	0	1	1	0
test 4:	0	0	1	0	0	1	0	1	0	1
test 5:	0	0	0	1	0	0	1	0	1	1

Figure 3: 1-cover-free matrix example.

3.1 CFF MATRIX CONSTRUCTIONS

Given d and n, we wish to find a d-CFF(t, n) for the smallest possible t, which we call $t(d, n)$. In Section 3.1.1 we present the optimal construction for $d = 1$. In Sections 3.1.2 and 3.1.3 we present two constructions for general d. The constructions presented here are well known in the literature; however, we provide more detailed proofs of this results. In Section 3.1.4 we show how to select the best construction for specific relative values of d and n.

3.1.1 The optimal construction for $d = 1$

In this section we give a CFF matrix construction for $d = 1$ with minimum t. Let us consider the vectors corresponding to the columns of the CFF matrix. Given a subset $S \subset \{1, \ldots, t\}$, the characteristic vector of S is a vector $x_S \in \{0, 1\}^t$ such that

$$(x_S)_i = \begin{cases} 1 & \text{if } i \in S \\ 0 & \text{otherwise} \end{cases}$$

The columns of the matrix can be seen as the characteristic vectors of subsets of $\{1, \ldots, t\}$. The definition of $1-$CFF matrix is equivalent to requiring that no column "is contained"in another column. More precisely, considering the subsets whose characteristic vectors are the columns of the matrix, no such subset is contained in any other. A collection of sets such that no set is contained in any other set is said to have the *Sperner property*. In other words, our problem is equivalent to finding the minimum t such that there exists a collection of subsets of a t-set with the Sperner property and with at least n such subsets.

Sperner theorem provides precise information to solve this problem.

Theorem 1. *(SPERNER, 1928) Let $\mathcal{A}$ be a collection of subsets of $\{1,\ldots,t\}$ such that $A_1 \not\subseteq A_2$ for all $A_1, A_2 \in \mathcal{A}$. Then $|\mathcal{A}| \leq \binom{t}{\lfloor t/2 \rfloor}$. Moreover, equality holds when $\mathcal{A}$ is the collection of all the $\lfloor t/2 \rfloor$-subsets of $\{1,\ldots,t\}$.*

Corollary 1. *Given n and $d = 1$, the smallest number t of tests possible in a $1 - CFF(t,n)$ matrix is $t(1,n) = \min\{t : \binom{t}{\lfloor t/2 \rfloor} \geq n\}$.*

Using Stirling formula, it is possible to approximate the value of $t(1,n)$, as given in the next proposition.

Proposition 2. *$t(1,n) \sim \log_2 n$ as $n \to \infty$.*

In conclusion, to build a CFF matrix with minimum number of rows t, for given n and $d = 1$ we use the following steps:

1. Calculate $t = t(1,n) = \min\{t : \binom{t}{\lfloor t/2 \rfloor} \geq n\}$.
2. List n distinct $\lfloor t/2 \rfloor$-subsets of $\{1,\ldots,t\}$ and use their characteristic vectors as the columns of the matrix.

As an example, for $n = 10$ we obtain $t(1,10) = 5$ and use all the 2-subsets of $\{1,2,3,4,5\}$:

$$\{\{1,2\},\{1,3\},\{1,4\},\{1,5\},\{2,3\},\{2,4\},\{2,5\},\{3,4\},\{3,5\},\{4,5\}\}$$

yielding the matrix in Figure 3.

3.1.2 A construction with $(d+1)\sqrt{n}$ tests

Pastuszak et al. (PASTUSZAK; PIEPRZYK; SEBERRY, 2000) construct matrices with the cover-free property from mutually orthogonal Latin squares (MOLS), which we present here. A Latin square of order s is an $s \times s$ matrix with s different elements (or symbols) in which each element occurs only once in each row and column. Two Latin squares are orthogonal (MOLS) if their ordered pairs of elements occur exactly once in each cell when we superimpose both squares. A set of mutually orthogonal Latin squares (MOLS) is a set of Latin square such that any two squares are orthogonal.

The construction of the CFF matrix works as follows. Considering d mutually orthogonal Latin squares of order s: $L = \{L_1,\ldots,L_d\}$

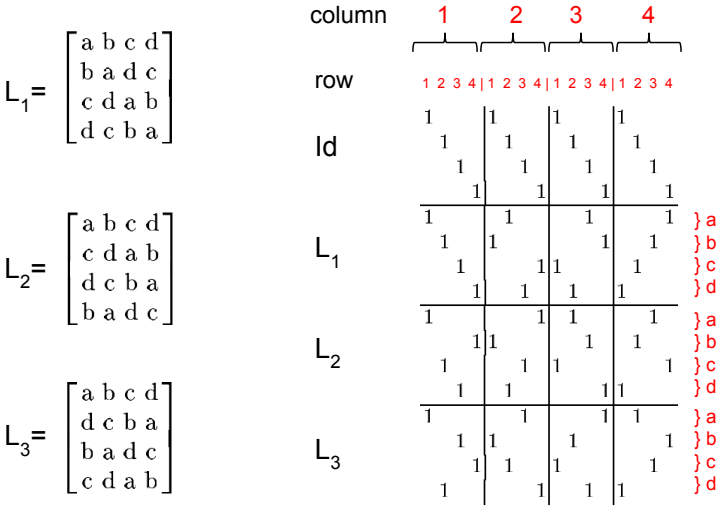

Figure 4: Example of matrix constructed using PPS, adapted from (PASTUSZAK; PIEPRZYK; SEBERRY, 2000).

and its elements $x = \{x_1, x_2, \ldots, x_s\}$. Our matrix M is composed by smaller $s \times s$ submatrices $M^{i,j}$, for $0 \leq i \leq d$ and $1 \leq j \leq s$. The row $M^{0,j}$, $1 \leq j \leq s$, is composed by $s \times s$ identity matrices. Each matrix $M^{i,j}$ represents the positions of all elements x at column j of the Latin square L_i, for $1 \leq i \leq d$ and $1 \leq j \leq s$ (see an example in Figure 4). Each row e of $M^{i,j}$ corresponds to element x_e, $1 \leq e \leq s$, and each column f of $M^{i,j}$ corresponds to row f of L_i, $1 \leq f \leq s$; more precisely, the construction of each matrix $M_{i,j}$ is given by:

$$(M^{i,j})_{e,f} = \begin{cases} 1, & \text{if } (L_i)_{f,j} = x_e \\ 0, & \text{otherwise.} \end{cases}$$

At the end we will have a CFF matrix with a special property that allows the gradual increment of d, reusing the previous tests if the first guess of d was not correct. In this way, the first $(d+1)s$ tests are executed and as d increases, s new tests (rows of the matrix) are executed, where the possible values of d are $d = 1, 2, \ldots, s-1$.

Now we give some results in order to show that the matrix M is cover-free. In the rest of this section, we will abuse notation and use the same notation c for a column of M and for the subset of $\{1, \ldots, (d+1)s\}$ that has this column as its characteristic vector. In this way, we can

talk about the intersection of two columns.

Lemma 1. *Given any distinct columns c_1 and c_2, then $|c_1 \cap c_2| \leq 1$.*

Proof. Let us index a column c_i of M by (f_i, j_i) where c_i comes from column j_i and row f_i of the Latin squares. So, $c_1 = (f_1, j_1)$ and $c_2 = (f_2, j_2)$. We separate the proof into the following cases.

Case 1: $j_1 = j_2$
We claim that $|c_1 \cap c_2| = 0$. Indeed, if we had non-empty intersections, it would mean that some of the Latin squares L_i have the same element x_e repeated in the same column j_1, which is not possible due to the definition of Latin squares.

Case 2: $f_1 = f_2$
We claim that $|c_1 \cap c_2| = 1$. Indeed, the intersection comes from matrices M^{0,j_1} and M^{0,j_2} in position (f_1, f_1). If we had intersection larger than one, it would mean that some of the Latin squares have the same element x_e repeated in the same row f_1.

Case 3: $f_1 \neq f_2$, $j_1 \neq j_2$
We claim that $|c_1 \cap c_2| = 1$. Indeed, if we had intersection larger than one, it would mean that, besides the element x_e in positions $(L_i)_{f_1,j_1}$ and $(L_i)_{f_2,j_2}$ (the first intersection) of the Latin square L_i, we would also have element $x_{e'}$ in positions $(L_j)_{f_1,j_1}$ and $(L_j)_{f_2,j_2}$ of Latin square L_j. This would force the pair $(x_e, x_{e'})$ to appear twice in the superimposition of L_i and L_j, contradicting the fact that they are orthogonal. □

Theorem 2. *(PASTUSZAK; PIEPRZYK; SEBERRY, 2000) The $(d+1)s \times s^2$ matrix M is cover-free.*

Proof. In order to guarantee that the matrix M is cover-free, we need to show that any column c is not covered by the union of any other set of d columns $C = \{c_1, \ldots, c_d\}$. By Lemma 1, we have $|c \cap c_i| \leq 1$, for $1 \leq i \leq d$, which implies

$$|c \cap (c_1 \cup \ldots \cup c_d)| \leq |c \cap c_1| + |c \cap c_2| + \ldots + |c \cap c_d| \leq 1 + \ldots + 1 = d.$$

To show that the matrix is cover free, it is enough to prove that the cardinality of c is greater than d. Since each $M^{i,j}$ has precisely one 1 per column, each column of M has precisely $d + 1$ ones, completing the proof.

□

Now we states this results in terms of number of elements n.

Corollary 2. *Let $n = s^2$. Then, the construction of matrix M yields a d-CFF$((d+1)\sqrt{n}, n)$.*

3.1.3 A construction with $(d+1)^2 \ln n$ tests

In this section, we present a construction of a strongly selective family (SSF) based in Error Correcting Codes (ECC) with large distances given, by Porat and Rothschild (PORAT; ROTHSCHILD, 2011). As we will see in this section, SFF matrices yield CFF matrices. An error-correcting code $\mathcal{C}$ consists of q^k codewords of length m over an alphabet of q letters and have minimum Hamming distance $D = \delta m$, $0 < \delta \leq 1$, which represents the minimum number of positions that two codewords differ, and can be denoted as $(m, k, \delta m)_q - ECC$. A linear code is an error-correcting code over the alphabet $\mathbb{F}_q$, and they have a $m \times k$ generator matrix $\mathcal{G}$ that generates the codewords by combining the words as $\mathcal{C} = \{\mathcal{G}y | y \in \mathbb{F}_q^k\}$. A family $\mathcal{F}$ is composed by subsets of $\{1, \ldots, n\}$ and it is known as strongly selective $(n, d)-$SSF if any of its elements is selected out of a small subset. That is, an element x is selected out of A by a subset B if $A \cap B = \{x\}$. A family of subsets $\mathcal{F}$ is strongly selective if each element in every subset $A \subseteq \{1, \ldots n\}$ with $|A| = d$ is selected by a subset of $\mathcal{F}$. SSFs and CFFs are closely related: an $(n, d+1)-$SSF with t subsets is a d-CFF(t, n), as shown later in Theorem 4.

The idea behind the method is that given a set of codewords from the ECC and another codeword w, there will be positions in w that differs from all the set of codewords. In other words, we can isolate a codeword from the others by the differing positions, which is similar to a CFF where any column is not covered by up to d other columns (representing the invalid elements). It is possible to construct a $(d+1)^2 \ln n \times n$ CFF matrix in $\mathrm{O}((d+1)n \ln n)$ time based on the following construction.

Consider $\mathcal{C} = \{w_1, \ldots, w_n\}$ an $(m, \log_q n, \delta m)_q-$ECC with $\delta < 1$. The SSF will be constructed from the codewords that have the same letter v in the same position p. More specifically, for each letter v and for each position p, $s_{p,v} = \{i \in \{1, \ldots, n\} | w_i[p] = v\}$ and $\mathcal{F}(\mathcal{C}) = \{s_{p,v} | p \in \{1, \ldots, m\}$ and $v \in \{1, \ldots, q\}\}$. At the end we will have $(d+1)^2 \ln n$ sets of codeword indexes in $\mathcal{F}(\mathcal{C})$ that form the SSF, and each set of indexes will determine the elements presented in each test of the CFF.

Theorem 3. *(PORAT; ROTHSCHILD, 2011) The constructed $\mathcal{F}(\mathcal{C})$ is an $(n, \frac{1}{1-\delta})$-SSF.*

Proof. Let $d = \frac{1}{1-\delta}$. In order to prove that $\mathcal{F}(\mathcal{C})$ is SSF, we need to show that given any set of d codewords $\{w_{i_1}, \ldots, w_{i_d}\}$, word w_{i_1} is selected from $\{w_{i_1}, \ldots, w_{i_d}\}$ by $\mathcal{F}(\mathcal{C})$. We know that for any $j \neq 1$, the maximum number of positions $p \in \{1, \ldots, m\}$ where $w_{i_1}[p] = w_{i_j}[p]$ is $(1-\delta)m$. This is because the minimum distance between any two codewords of this ECC is, following its construction, δm, and if this is the number of positions in which the two codewords of length m differ, these words can have at most $m - \delta m = (1-\delta)m$ positions with the same letter. Therefore, the number of positions where $w_{i_1}[p] \in \{w_{i_2}[p] \ldots, w_{i_d}[p]\}$ is at most $(d-1)(1-\delta)m$. Since $d = \frac{1}{1-\delta}$, we can see that $(d-1)(1-\delta)m < m$, and consequently there exist at least one position where $w_{i_1}[p] \notin \{w_{i_2}[p] \ldots, w_{i_d}[p]\}$. In other words, there is a subset $s_{p,v}$ where $i_1 \in s_{p,w_{i_1}[p]}$ while all the other i_j-s are not, which means that i_1 is selected by $s_{p,w_{i_1}[p]}$, which is one of the sets that form $\mathcal{F}(\mathcal{C})$. □

Theorem 4. *(PORAT; ROTHSCHILD, 2011) A $(n, d+1)-$SSF is a d-CFF(t, n).*

Proof. The $(n, d+1)-$SSF is composed by subsets of indexes $s_{p,v}$, where each index $i \in \{1, \ldots, n\}$ represents a codeword w_i. The strongly-selective property guarantees that given any subset $A \subseteq \{1, \ldots, n\}$ with $|A| = d+1$, each codeword c of A is selected out by some subset B of the SSF, where $A \cap B = \{c\}$. Consider that each column of the CFF is represented by a codeword w_i and each row of the CFF is a characteristic vector of each subset $s_{p,v}$. In order to guarantee that $(n, d+1)-$SSF is a d-CFF(t, n), we need to show that any column c is not covered by the union of any other set of d columns $C = \{c_1, \ldots, c_d\}$. If we consider $A = C \cup \{c\}$ as a set of $d+1$ columns, by the strongly selective property we guarantee that exists a subset B of the SSF that selects c, $A \cap B = \{c\}$ which implies that $C \cap B = \emptyset$ and c belongs to B.

□

In order to construct a d-CFF(t,n), with $t = (d+1)^2 \ln n$, we will construct an $(n, d+1)$-SSF with the same t. Consider an $m \times k$ generator matrix $\mathcal{G}$ of a linear code and a minimum distance δm. To build such a linear code, we need to assure that the distance (or weight) of each codeword $x = \mathcal{G}y$ satisfies $\omega(x) \geq \delta m$. For any generator matrix $\mathcal{G}$, we define a goal function $goal(\mathcal{G})$. For a boolean condition B, let χ be an

Algorithm 1 Algorithm that computes a d-CFF(t, n) with $t = \Theta((d+1)^2 \ln n)$

Input: n, d
Choose q to be a prime power such that $q \in [2(d+1), 4(d+1))$;
$k = \log_q n$; $\delta = \frac{d}{d+1}$;
$m = \frac{k}{1-H_q(\delta)}$, where $H_q(\delta) = \delta \log_q \frac{q-1}{\delta} + (1-\delta)\log_q \frac{1}{1-\delta}$;
Initialize $\mathcal{G}$ as an $m \times k$ matrix;
foreach $i \in \{1, \ldots, m\}$
    foreach $j \in \{1, \ldots, k\}$
        Choose a value for $\mathcal{G}[i, j]$ in $\mathbb{F}_q$ in order to minimize the expected value of $goal(\mathcal{G})$ given all the values of j chosen so far;
Output: The binary $mq \times n$ matrix corresponding to the linear code generated by $\mathcal{G}$.

indicator of this condition, that is $\chi(B) = 1$ if B is true and $\chi(B) = 0$ if B is false. We define the goal function $goal(\mathcal{G}) = \sum_{y \in \mathbb{F}_q^k \neq 0} \chi(\omega(\mathcal{G}y) < \delta m)$. Note that if $goal(\mathcal{G}) = 0$, then the linear code has minimum distance at least δm.

One possible algorithm to build such a code consists of repeatedly choosing the entries of $\mathcal{G}$ randomly until we achieve $goal(\mathcal{G}) = 0$. Porat and Rothschild (PORAT; ROTHSCHILD, 2011) derandomizes this probabilistic algorithm by determining the entries of $\mathcal{G}$ one by one, while trying to minimize $goal(\mathcal{G})$ as given in Algorithm 1. They prove that the algorithm will result in a code with $goal(\mathcal{G}) = 0$ and with this code it is possible to construct an SSF in time $\theta((d+1)n \log n)$.

3.1.4 Using the best constructions for d and n

To use a d-CFF(t, n) matrix that is efficient for our applications we need to minimize t which both determines the number of aggregated signatures required and the speed of the verification, as will be seen in the next chapters. Another important variable is the number w of 1's in the matrix, as the speed of encoding and decoding is also affected by it. We summarize below the existing constructions that are used to obtain the upper bounds on number t of tests and number w of 1's in the CFF matrix. Here we consider as *S1* the construction generated by Sperner's theorem (Section 3.1.1), *PPS* the construction by Pastuszak et al. (Section 3.1.2), *PR* the construction by Porat and Rothschild

(Section 3.1.3) and I_n the use of the identity matrix.

Table 1: Values of t and w for each CFF construction.

methods	t	w
S1	$\sim \log_2 n$	$\sim \lfloor \frac{\log_2 n}{2} \rfloor n$
PR	$(d+1)^2 \ln n$	$\frac{(d+1)}{2} n \ln n$
PPS	$(d+1)\sqrt{n}$	$n(d+1)$
I_n	n	n

The values of t comes from the number of rows of each construction.

For $d = 1$, Sperner theorem gives $t = \min\{s : \binom{s}{\lfloor s/2 \rfloor} \geq n\}$ and $t \to \log_2 n$ as $n \to \infty$. Since each column of the matrix has $\lfloor t/2 \rfloor$ ones and the matrix has n columns, we obtain $w \approx \lfloor \frac{t}{2} \rfloor n \approx \lfloor \frac{\log_2 n}{2} \rfloor n$.

We obtain the values w and t for construction PR as follows. We know that each column of the matrix has m ones, since it represents a codeword with length m, and we have n columns. So, the total number w of ones in the matrix is $w = mn$. We also know that $t = mq$ and $q \geq 2(d+1)$. Considering this, we have:

$$w = mn = \frac{t}{q} n \leq (d+1)^2 \ln n \frac{n}{2(d+1)} = \frac{(d+1)}{2} n \ln n$$

The values of w and t in construction PPS come directly from Definition 3 in (PASTUSZAK; PIEPRZYK; SEBERRY, 2000), where since we have $(d+1)$ ones per column and a total of n columns, it gives us a matrix with a total of $n(d+1)$ ones.

We now have a criteria to select among the various existing constructions of CFF matrices, depending on the relative values of d and n, in order to obtain the smallest t. The next two chapters, propose methods that require the generation of CFF matrices given here.

4 LOCATION OF MODIFICATIONS IN SIGNED DOCUMENTS

The content of this chapter is based on an article published in Information Processing Letters (IDALINO et al., 2015).

4.1 INTRODUCTION

Digital signature schemes can detect if modifications were done in a signed document, but do not offer information on where exactly those modifications occurred. In this context, even a single bit change would invalidate the whole document. In the present chapter, we provide a general Modification Location Signature Scheme, which determines which parts of the document were modified, thus ensuring partial data integrity.

Partial data integrity is useful in several scenarios. First, we may need to ensure the integrity of specific parts of a document. For example, in fillable forms the owner may need to assure that the document is official, while some parts are expected to be modified. Second, in a data forensics investigation of a crime, the investigator could have more clues on who is the attacker by knowing what exactly was modified (GOODRICH; ATALLAH; TAMASSIA, 2005). Third, assuring that part of the data is intact can improve the efficiency of a computer system. For example, in a large database, the modification of some of its records would not invalidate the whole database, avoiding total disruption of service.

One can also see partial data integrity as a solution for guaranteeing privacy protection, where the extraction of selected portions of a signed document is to be shared with another party (content extraction signature (STEINFELD; BULL; ZHENG, 2002), redactable signature (JOHNSON et al., 2002)). Our signature scheme capable of locating modifications can be used in this application by substituting the removed parts by "blank"symbols. The original signature can be used to guarantee the integrity of the non-removed parts.

The Modification Location Signature Scheme (MLSS) proposed in this paper employs combinatorial group testing to determine which blocks of a document contain modifications and which ones are intact. This work is closely related to the work of Zaverucha and Stinson (ZAVERUCHA; STINSON, 2009) who propose the use of group testing to

identify modified documents in batch. However, while in (ZAVERUCHA; STINSON, 2009) group testing is used on the verifier's end to speed up the batch verification algorithm, in our approach it is used both at the signer's and verifier's end, which greatly improves the signature size over the trivial idea of treating each block of a document as an independent signed document. This trivial idea would require n signatures for a document divided into n blocks, which would not be efficient, while for the cases of interest here we would have the size of a signature multiplied by a factor of $O(\log n)$ instead (see Theorem 5 and the discussion that follows it). While MLSS is applicable to any type of document (text, pictures, videos or a mix), the type of document may influence the way one divides it (see Section 4.7).

4.2 DEFINITION OF THE PROBLEM AND RELATED WORK

Following a general definition (ZAVERUCHA; STINSON, 2009), a signature scheme is specified by algorithms (GEN, SIGN, VERIFY). GEN(k) receives a security parameter k and outputs a pair of keys (s_k, p_k), a secret key used for signing and a public key used for verification, respectively. SIGN(s_k, m) outputs a signature σ on the message m using the secret key s_k. VERIFY(p_k, σ, m) outputs 1, using the public key p_k, if σ is a valid signature of m, and 0 otherwise.

We propose a general digital signature scheme for signing a document divided into blocks providing, in the case of modifications on the document after signing, the extra capability of locating which blocks have been modified.

Definition 2. *Modification Location Signature Scheme (MLSS):*
Let $B = (B_1, \ldots, B_n)$ be a document divided into n blocks. MLSS-GEN*(k) receives a security parameter k and outputs a pair of keys (s_k, p_k).* MLSS-SIGN*(s_k, B) outputs a signature σ on B using the secret key s_k.* MLSS-VERIFY*(p_k, σ, B) outputs 1 if, using the public key p_k, σ is a valid signature of B, it outputs 0 if σ has been modified or is not authentic, and otherwise (B has been modified) outputs extra information on the location of the modifications in B.*

Here we present an approach, which is based on combinatorial group testing, to solve the challenge stated in Definition 2. Zaverucha and Stinson (ZAVERUCHA; STINSON, 2009) observe that finding invalid signatures in a batch is a group testing problem, and propose the use of group testing methods to improve the efficiency of the batch verification algorithm. In (ZAVERUCHA; STINSON, 2009), by exploring the

best known group testing algorithms, they show how to run t signature verifications to verify a batch of n signed documents, where t is substantially smaller than n; in their case, adaptive and nonadaptive group testing can be used. We use a similar idea to improve the verification algorithm but, while in (ZAVERUCHA; STINSON, 2009) group testing is used on the verifier's end to speed up the batch verification algorithm, in our approach it is used both at the signer's and verifier's end in order to minimize the signature size.

We produce t digests, each one involving a subset of the n blocks of the document (t much smaller than n, see Theorem 5 and the discussion that follows it). This tuple of digests is signed and sent with the document, allowing the verifier to determine the blocks that were modified. This approach requires nonadaptive group testing, since the digests must be prepared at the signer's end independently of where modifications may occur. In this case, we need an upper bound d on the number of modified blocks; this threshold value d needs to be chosen carefully to keep control on the size t. Notice that MLSS is applicable to any type of document (text, pictures, videos or a mix), but the type of document may influence the way one divides it (see Section 4.7). We consider that the security of the method relies on the security of the signature algorithms and hash functions used.

4.3 METHOD AND ALGORITHMS

In our scheme, both signer and verifier use the same $t \times n$ matrix for a d-cover-free family via a call to a deterministic function MLSS-CFF(d, n), which can use constructions presented in Proposition 4, given in Section 4.4. We add a parameter d in our algorithm, built on a given (traditional) signature scheme (GEN, SIGN, VERIFY). Algorithm MLSS-GEN(k) consists of a simple call to GEN(k).

Considering a document divided into blocks $B = (B_1, B_2, .., B_n)$, algorithm MLSS-SIGN works as follows. Let $h_1, h_2, \ldots, h_n$ be the result of a public hashing algorithm $h(\cdot)$ applied on blocks $B_1, B_2, \ldots, B_n$, respectively. The tests, given by each row i of the matrix, indicate which block hashes h_j of the document are to be concatenated to form a test T_i which is a digest of these concatenated hashes. Another digest $h^* = h(B)$ is calculated from B yielding $T = (T_1, T_2, \ldots, T_t, h^*)$. The signature of B is given by $\sigma = (T, \sigma')$, where $\sigma' = \text{SIGN}(s_k, T)$. We note that T needs to be part of the signature since otherwise we would not be able to identify the modified blocks, as we present next.

At the receiver end, the MLSS-VERIFY algorithm verifies if σ is a valid signature by verifying T with σ'. If so, then it compares h^* with the hash of the received document B'. If they match, the document was not modified; otherwise, the algorithm locates the modified blocks as follows. Using the same method as the sender, it computes $(T'_1, T'_2, \ldots, T'_t)$ from B'. The set of indexes i where $T'_i \neq T_i$ indicates which tests have failed and using group testing, it deduces exactly which blocks B_j have been modified. This process is depicted in Figure 5. Algorithm MLSS-VERIFY also allows a faster verification that does not locate the modified blocks, much as a standard verification algorithm; this option is applied by setting the boolean location parameter lc to false. The signature and verification algorithms are given next.

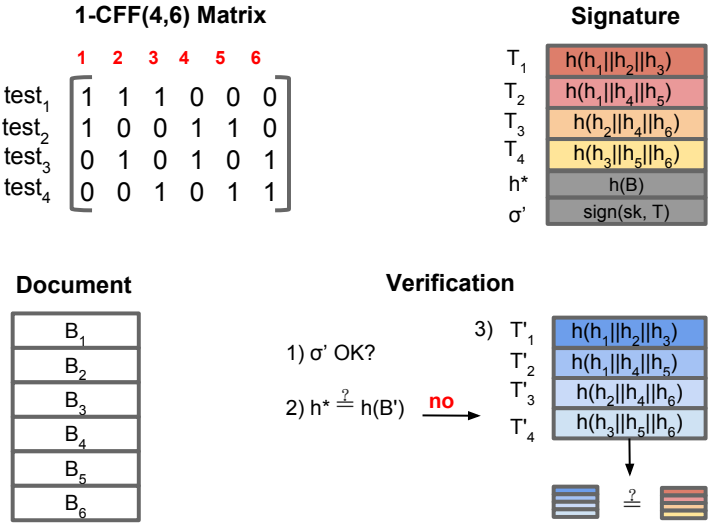

Figure 5: MLSS signature and verification scheme.

MLSS-SIGN(s_k, B, d)
Input: secret key s_k, document $B = (B_1, B_2, \ldots, B_n)$, modification threshold d.
Output: a signature σ.

1. Use n and d to determine t and the d-CFF(t, n) matrix M to be used:

$M = \text{MLSS-CFF}(d, n)$.

2. Let $h_j \leftarrow h(B_j)$, $1 \leq j \leq n$. Use $(h_1, h_2, \ldots, h_n)$ and M to compute $T_1, \ldots, T_t$, as follows: for each row $1 \leq i \leq t$ compute c_i, which is the concatenation of the hashes h_j for j such that $M_{i,j} = 1$, and let $T_i = h(c_i)$. Calculate $h^* = h(B)$ and set $T = (T_1, T_2, \ldots, T_t, h^*)$.

3. Compute $\sigma' = \text{SIGN}(s_k, T)$ and output $\sigma = (T, \sigma')$.

The verification algorithm has three possible outcomes: signature has been modified (output 0); signature and document were not modified (output 1); signature was not modified and document has been modified (output I as the set of modified block indices, if $lc =$ true; output 2, otherwise).

MLSS-VERIFY(p_k, σ, B', d, lc)
Input: public key p_k, signature σ, document $B' = (B'_1, B'_2, \ldots, B'_n)$, modification threshold d, boolean location parameter lc.
Output: 0, 1, 2 or I, corresponding to the outcomes above explained.

1. Verify σ: Let $\sigma = (T, \sigma')$ and $T = (T_1, T_2, \ldots, T_t, h^*)$. If $\text{VERIFY}(p_k, T, \sigma') = 0$ then output 0 and exit.

2. Compute $h^{**} \leftarrow h(B')$. If $h^* = h^{**}$ then output 1 and exit.

3. If $lc =$ false, then output 2 and exit.

4. Use n and d to determine the d-CFF(t, n) matrix M to be used: $M = \text{MLSS-CFF}(d, n)$.

5. Use the same process as Step 2 of MLSS-SIGN to compute $T' = (T'_1, \ldots, T'_t)$ from B' using M and $h(\cdot)$.

6. Compare T and T' and record the discrepancies (failing tests): $F = \{i \in \{1, \ldots, t\} : T_i \neq T'_i\}$.

7. Use group testing to determine the modified blocks:
   Initialize $I = \{1, \ldots, n\}$;
   for each $i \notin F, 1 \leq i \leq t$, do
   for each $j \in I$ such that $M_{i,j} = 1$ do $I \leftarrow I \setminus \{j\}$;
   output I.

Remark 1. *If the number of modified blocks is larger than d, then the algorithm outputs a set I, with $|I| > d$, that contains all the modified blocks and possibly some more. However, any block that is not in I was not modified.*

4.4 CORRECTNESS AND COMPLEXITY

Proposition 3. *Consider a document and its signature generated by* MLSS-SIGN. *Then, algorithm* MLSS-VERIFY *correctly verifies the signature according to the three possible outcomes. In particular, if the signature is valid and there are up to d modifications, then I is precisely the set of indices of these modified blocks.*

Proof. Steps 1 and 2 of MLSS-VERIFY identify the case of invalid signature and the case of valid signature with unmodified document, respectively. The next steps deal with the case of valid signature and modified document. Depending on the boolean parameter lc, the algorithm exits at Step 3 or move on to locate the modified blocks. Steps 4 to 6 perform the hash concatenations dictated by matrix M to reproduce the creation of tests $(T'_1, \ldots, T'_t)$ from B'. The correctness of Step 7 of MLSS-VERIFY follows directly from the properties of a cover-free family, and the assumption that the hash function used has the desired property (no collisions). A matching test $T_i = T'_i$ guarantees that all the blocks that are concatenated to produce T_i have not been modified. If the total number of modified blocks is at most d, then every unmodified block B_j is part of some matching test T_i, and the remaining blocks are precisely the modified blocks. □

We now analyze the algorithms and compare them with traditional signature schemes. Denote by $comp(x)$ the cost of comparing x bits. Let b be the size of B in bits, w be the number of 1's on the CFF matrix M, and t be the number of rows in M. Denote by $cost_{CFF}(d, n)$ the cost of computing function MLSS-CFF(d, n).

Theorem 5. *Consider* MLSS-SIGN *and* MLSS-VERIFY *algorithms. Assume that the cost (running time) of computing the hash function h, denoted by* $cost_h$, *is a linear function on the input size, and let* h_{out} *denote the number of bits of the output of h. Assume algorithm* SIGN (VERIFY) *first applies function h on its input message and then applies a signature method (verification method) with cost denoted by* $cost_{sign}(h_{out})$ $(cost_{verify}(h_{out}))$. *Then,*

1. *The size of the signature* σ *produced by* MLSS-SIGN *is* $(t+1)h_{out} + |\sigma'|$ *while the size produced by a traditional signature method is* $|\sigma'|$.

2. *The running time of* MLSS-SIGN *is* $cost_h(2b + (w + t + 1)h_{out}) + cost_{sign}(h_{out}) + cost_{CFF}(d, n)$, *while the running time of* SIGN *is*

$cost_h(b) + cost_{sign}(h_{out})$.

3. *The running time of* MLSS-VERIFY *when the signature is invalid (output 0), or the document has not been modified (output 1) is* $cost_h(b + (t+1)h_{out}) + cost_{verify}(h_{out}) + comp(h_{out})$ *(this is also the cost when* $lc = false$*); when the document has been modified but the signature has not and* $lc = true$*, the running time of* MLSS-VERIFY *is* $cost_h(2b + (w+t+1)h_{out}) + cost_{verify}(h_{out}) + cost_{CFF}(d,n) + comp((t+1)h_{out}) + c.w$*, where* c *is a constant. The running time of* VERIFY *is* $cost_h(b) + cost_{verify}(h_{out})$.

Proof. The size of the signature σ comes directly from its form, composed by $t+1$ hashes T and the signature σ'. In Step 1, MLSS-SIGN computes the matrix M; in Step 2 it computes n hashes with total input size b, followed by t hashes of total input size $w \cdot h_{out}$ plus a hash of the entire document with input size b; and in Step 3 we have assumed that SIGN applies a hash on T, which has size $(t+1)h_{out}$. Hence, the linearity of $cost_h$ yields $cost_h(2b + (w+t+1)h_{out})$ for hash computations plus the cost of signing a message of size h_{out} and computing matrix M. If we apply algorithm SIGN directly to the message B we have $cost_h(b) + cost_{\text{sign}}(h_{out})$, instead.
In MLSS-VERIFY, Step 1 yields the cost of VERIFY of an input of size $(t+1)h_{out}$, and Step 2 uses a hash of the whole document plus a comparison of h_{out} bits, giving $cost_h(b + (t+1) \cdot h_{out}) + cost_{\text{verify}}(h_{out}) + comp(h_{out})$. In the case of blocks that have been modified with a valid signature and lc = true, the cost incurred by Steps 4 to 6 is $cost_h(b + w \cdot h_{out}) + comp(t \cdot h_{out}) + cost_{CFF}(d,n)$. Step 7 can be done in time linear with w. □

We now discuss practical implications of Theorem 5. The most significant cost in digital signature algorithms is related to the use of cryptographic functions. If we compare our algorithms with the traditional ones, we note that this cost remains the same, while most of our extra cost comes from additional hash computations. In Section 4.5, we give a detailed comparison of these algorithms based on a standard digital signature method. As an illustration, for a document divided into $n = 256$ blocks with 1024 bytes per block and $d = 10$, the running time of MLSS-SIGN (the calculation of $cost_h$ and $cost_{sign}$) is 4.8561 ms, while a traditional SIGN costs 2.8804 ms. If the document is not modified or the signature is invalid or lc = false, the running time of MLSS-VERIFY is 1.5246 ms while VERIFY is 1.4934 ms. If there are modifications but the signature is valid and lc = true, we locate

$d = 10$ modified blocks using a total running time of 3.4691 ms. More experiments are provided in Section 4.5.

We note that the signature σ generated by MLSS-SIGN has an additional size of $(t+1)h_{out}$ bits. In order to keep $|\sigma|$ as small as possible we need to minimize t, the number of rows in the CFF matrix. To obtain a small enough t, we consider constructions of Sperner (SPERNER, 1928) (S1), Porat and Rothschild (PORAT; ROTHSCHILD, 2011) (PR), Pastuszak et al. (PASTUSZAK; PIEPRZYK; SEBERRY, 2000) (PPS), as well as using identity matrix I_n; we use the relatively better t given d and n to choose one construction among these, as given in Proposition 4. A small t is also important to reduce the extra computation costs for signing and verifying; moreover, given t it is desirable to choose a CFF matrix with the smallest w.

Proposition 4. *Let d, n be integers, and let t be the number of rows and w be the number of ones of the CFF matrix given by function* MLSS-CFF(d, n). *Then the table below gives the values of t and w, for each the four constructions specified.*

ranges of d, n	t	w
$d = 1$, *for all* n: *use* S1	$\sim \log_2 n$	$\sim \lfloor \frac{\log_2 n}{2} \rfloor n$
$d \in [2, \frac{\sqrt{n}}{\ln n})$: *use* PR	$(d+1)^2 \ln n$	$\frac{(d+1)}{2} n \ln n$
$d \in [\frac{\sqrt{n}}{\ln n}, \sqrt{n} - 1)$: *use* PPS	$(d+1)\sqrt{n}$	$n(d+1)$
$d \geq \sqrt{n} - 1$: *use* I_n	n	n

Proof. The values of t and w are derived in Section 3.1.4. □

4.5 EXPERIMENTAL RESULTS

Here we provide some experiments using a standard digital signature (SHA256 with RSA 2048 bits) and *openssl*. We quantify the cost of computing $cost_h$, while ignoring other less relevant linear costs ($comp(.)$, $c \cdot w$) and the cost of CFF matrix computation, which could be preprocessed for specific applications. Indeed, we experimentally verify[1] that hash computations behave in *openssl* as a linear function of the input size (which is linear in t and w), approximately as $5.52 \times 10^{-9}x + 3.819 \times 10^{-7}$, agreeing with the assumptions of Theorem 5.

In the following tables we give running times obtained experimentally for a set of chosen values of the document size b, the number

[1] in an iMac 2.7 GHz Intel Core i5 with 6 MB on-chip L3 cache.

of blocks n and the number of tests t, where the costs are given in milliseconds and the document size b in bytes. In the tables below, "MLSS-S" refers to the running time of MLSS-Sign, "MLSS-V.1" refers to the running time of MLSS-Verify when there is no modifications or the signature is invalid or lc = false, while "MLSS-V.2" gives MLSS-Verify running time when modifications occurred and lc = true. Different tables use block sizes of 1024 and 8192 bytes.

Table 2: Comparison between methods with blocks of size 8192 bytes and $d = 1$.

b	n	t	w	Sign	MLSS-S	Verify	MLSS-V.1	MLSS-V.2
16,384	2	2	2	1.5238	1.6151	0.1368	0.1373	0.2281
65,536	8	5	20	1.7951	2.1614	0.4081	0.4092	0.7744
262,144	32	7	112	2.8804	4.3486	1.4934	1.4948	2.9616
1,048,576	128	10	640	7.2215	13.1246	5.8345	5.8364	11.7376

Table 3: Comparison between methods with blocks of size 1024 bytes and $d = 1$.

b	n	t	w	Sign	MLSS-S	Verify	MLSS-V.1	MLSS-V.2
16,384	16	6	48	1.5238	1.6239	0.1368	0.1380	0.2369
65,536	64	8	256	1.7951	2.2037	0.4081	0.4097	0.8167
262,144	256	11	1408	2.8804	4.5782	1.4934	1.4955	3.1912
1,048,576	1024	13	6656	7.2215	14.1878	5.8345	5.8369	12.8008

Table 4: Comparison between methods with blocks of size 8192 bytes and $d = 2$.

b	n	t	w	Sign	MLSS-S	Verify	MLSS-V.1	MLSS-V.2
16,384	2	2	2	1.5238	1.6151	0.1368	0.1373	0.2281
65,536	8	8	8	1.7951	2.1599	0.4081	0.4097	0.7729
262,144	32	16	96	2.8804	4.3475	1.4934	1.4965	2.9605
1,048,576	128	33	384	7.2215	13.0836	5.8345	5.8406	11.6966

Table 5: Comparison between methods with blocks of size 1024 bytes and $d = 2$.

b	n	t	w	Sign	MLSS-S	Verify	MLSS-V.1	MLSS-V.2
16,384	16	12	48	1.5238	1.6250	0.1368	0.1391	0.2380
65,536	64	24	192	1.7951	2.1952	0.4081	0.4125	0.8082
262,144	256	48	768	2.8804	4.4717	1.4934	1.5020	3.0847
1,048,576	1024	62	10646	7.2215	14.9014	5.8345	5.8457	13.5144

As we can see in Tables 2 to 9, the running time of MLSS-Sign is on average the double of a traditional Sign for the biggest docu-

Table 6: Comparison between methods with blocks of size 8192 bytes and $d = 3$.

b	n	t	w	Sign	MLSS-S	Verify	MLSS-V.1	MLSS-V.2
16,384	2	2	2	1.5238	1.6151	0.1368	0.1373	0.2281
65,536	8	8	8	1.7951	2.1599	0.4081	0.4097	0.7729
262,144	32	22	128	2.8804	4.3542	1.4934	1.4975	2.9672
1,048,576	128	45	512	7.2215	13.1082	5.8345	5.8426	11.7212

Table 7: Comparison between methods with blocks of size 1024 bytes and $d = 3$.

b	n	t	w	Sign	MLSS-S	Verify	MLSS-V.1	MLSS-V.2
16,384	16	16	16	1.5238	1.6200	0.1368	0.1398	0.2330
65,536	64	32	256	1.7951	2.2079	0.4081	0.4139	0.8209
262,144	256	64	1024	2.8804	4.5198	1.4934	1.5048	3.1328
1,048,576	1024	110	14195	7.2215	15.5369	5.8345	5.8542	14.1499

Table 8: Comparison between methods with blocks of size 8192 bytes and $d = 10$.

b	n	t	w	Sign	MLSS-S	Verify	MLSS-V.1	MLSS-V.2
16,384	2	2	2	1.5238	1.6151	0.1368	0.1373	0.2281
65,536	8	8	8	1.7951	2.1599	0.4081	0.4097	0.7729
262,144	32	32	32	2.8804	4.3389	1.4934	1.4992	2.9519
1,048,576	128	124	1408	7.2215	13.2805	5.8345	5.8566	11.8935

Table 9: Comparison between methods with blocks of size 1024 bytes and $d = 10$.

b	n	t	w	Sign	MLSS-S	Verify	MLSS-V.1	MLSS-V.2
16,384	16	16	16	1.5238	1.6200	0.1368	0.1398	0.2330
65,536	64	64	64	1.7951	2.1796	0.4081	0.4196	0.7926
262,144	256	176	2816	2.8804	4.8561	1.4934	1.5246	3.4691
1,048,576	1024	352	11264	7.2215	15.0616	5.8345	5.8968	13.6746

ments. Otherwise, for the smaller documents, the running time can be similar to Sign (see columns Sign and MLSS-S). If the document is not modified or if the verifier is not interested in locating the modifications, the cost of MLSS-Verify is basically the same as a regular RSA Verify for all sizes of documents or values of d (see columns Verify and MLSS-V.1 for different values of d). Finally, the cost to locate d modifications with MLSS-V.2 remain the double of Verify even when the size of the document and the value of d increase for block sizes of 8192 bytes. For documents divided into more blocks (blocks of size 1024 bytes), the cost of MLSS-V.2 can be a little more than the

double, specially in the cases where the values of b and d are large (see columns VERIFY and MLSS-V.2 for different values of d).

In conclusion, for the values considered we observe a moderate increase in the running time of MLSS-SIGN and MLSS-VERIFY with respect to SIGN and VERIFY, and this increase is highest when n and d are larger. For the documents with $n = 1024$ the increase was by a factor between 2 and 2.5 when d varies from 1 to 10. We also remark that while this increase of time is always incurred for MLSS-SIGN, it is not incurred in MLSS-VERIFY if no modifications occurred or if the verifier does not wish to locate modifications (lc=false).

4.6 A VARIATION OF MLSS TO REDUCE THE SIGNATURE SIZE

Here we propose a variation of the MLSS scheme that was not included in (IDALINO et al., 2015).

In order to send even less data through network, we suggest a few modifications in the previous scheme. First we want to avoid sending t extra hashes, specially for the cases where the document was not modified or when the verifier is not interested in locating modifications, because in these cases the t hashes will not be used. In order to solve this problem we suggest to send only one hash of $(T_1, \ldots, T_t)$, instead of t hashes, just in order to verify σ'. Then, if the verifier still needs to localize the modifications, he can order later the t hashes separately. Before calculating T' and comparing with T, the verifier first verifies if the t hashes received where not modified by checking its hashes with h_t (which was previously verified in σ'). If the t hashes are intact, they can be used to locate the modified blocks as we can see in Figure 6.

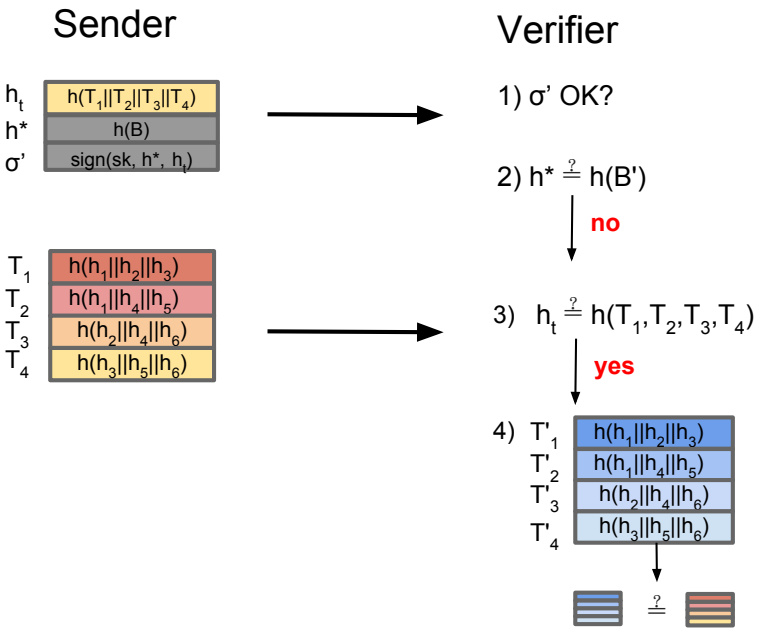

Figure 6: MLSS variation.

Theorem 6. *Considering this new variation of MLSS, the complexity behaves as follows:*

1. *The size of the signature produced by* MLSS-Sign *for the cases where the document was not modified or* $lc = false$ *is* $2 \times h_{out} + |\sigma'|$. *If the document was modified and* $lc = true$ *we need to send* t *extra hashes, with a total size of* $(t+2)h_{out} + |\sigma'|$

2. *The running time of* MLSS-Sign *is* $cost_h(2b + (w+t+2)h_{out}) + cost_{sign}(h_{out}) + cost_{CFF}(d,n)$.

3. *The running time of* MLSS-Verify *when the signature is invalid (output 0), or the document has not been modified (output 1) is* $cost_h(b + 2 \times h_{out}) + cost_{verify}(h_{out}) + comp(h_{out})$ *(this is also the cost when* $lc = false$*); when the document has been modified but the signature has not and* $lc = true$*, the running time of* MLSS-Verify *is* $cost_h(2b + (w+t+2)h_{out}) + cost_{verify}(h_{out}) + cost_{CFF}(d,n) + comp((t+2)h_{out}) + c.w$*, where* c *is a constant.*

Proof. The size of the signature presents $t-1$ less hashes than the previously proposed method since we now send only two extra hashes with σ' (h_t and h^*). In MLSS-SIGN we still computes n hashes with total input size b, followed by t hashes of total input size $w \cdot h_{out}$ plus a hash of the entire document with input size b. Now we additionally calculate a hash $h_t = h(T_1, \ldots, T_t)$ with input size of $t \cdot h_{out}$. At the end, SIGN applies a hash on h_t and h^*, which has size $2 \cdot h_{out}$. In MLSS-VERIFY, when the document is not modified or lc = false we verify the signature σ' by calculating a hash on h_t and h^* with an input of size $2 \cdot h_{out}$, plus a hash of the whole document plus a comparison of h_{out} bits. In the case of blocks that have been modified with a valid signature and lc = true, we additionally need to verify the t received hashes with h_t by calculating a hash with input size $t \cdot h_{out}$, then we calculate n hashes from the n blocks with input size b, plus t hashes of total input size $w \cdot h_{out}$ and $t+1$ comparisons, which gives us the $cost_h(2b + (w + t + 2)h_{out})$ and $comp((t+2)h_{out})$. □

With this modification we have an extra h_{out} in $cost_h$ for MLSS-SIGN and MLSS-VERIFY (if lc = true), but if the document was not modified or lc = false, we need to send $t-1$ less hashes and $(t-1)h_{out}$ less hash calculations (or $cost_h$ input). Additionally, this modification allows the possibility of choosing the location granularity. If we imagine a static document that will be send to several verifiers, the owner can store some versions of matrices and T's, taking into consideration different sizes of blocks and consequently different accuracies in the results. Then, the verifier who realized that the document was modified can order an specific granularity and receive more or less hashes depending on the choice. If the application can support more hash calculations, comparisons and bandwidth, its verifier can order a bigger t and consequently a more accurate location.

4.7 DIVISION IN BLOCKS AND BLOCK SIZES

An issue that needs to be considered is the scheme to divide the document into n blocks. Sender and receiver must use the same block organization, and we require that this organization must be preserved. For instance, dividing a text into blocks of the same size makes it hard to support modifications, as one bit inserted into block 1 would prevent us to keep track of where the other blocks are. Therefore, information on block structure must be included with the document (e.g. a description header) and legitimate modifications should be done

using a system that is "block aware".

We suggest two possible solutions to the problem of dividing the document into blocks. A first solution is to use special delimiters to separate blocks (e.g. tags on an XML document, or reserved characters on a text) or a description header that indicates where each block starts. A second solution is to use the own data organization to separate blocks (e.g. the records of a database). We could also use the semantics of the data to separate blocks for a specific application (e.g. sections of a document).

A second issue to be considered is the block size, which depends on the application needs and computational capabilities. The extreme values of block size may not be suitable. Block size equal to b ($n = 1$) means no ability to locate modifications, while very small block size makes n and t too large, rendering the scheme inefficient. We can observe the effect of number of blocks on running time for different values of d in Tables 2 to 9.

4.8 FINAL CONSIDERATIONS

In this chapter we propose a general Modification Location Signature Scheme (MLSS), where we decouple the verification of the signature from the possible modifications in the document. Our method is the first to address the issue of locating modified blocks. This is accomplished with some additional costs in hash computations but no additional costs in the cryptographic functions involved (SIGN and VERIFY).

5 LEVEL-D SIGNATURE AGGREGATION FOR OUTSOURCED DATABASES

The content of this chapter is based on a preprint of our paper with the same title as this chapter.

5.1 INTRODUCTION

In an outsourced database scenario, the data owners use a third-party service provider to manage their data. The outsourced database clients can store, update and query their data without worrying about installation and maintenance of the database.

There are three models of outsourced databases: unified client model, multiquerier model and multiowner model. The former consists in a database that is used by a single user, i.e., the data owner; in a multiquerier model other clients (queriers) besides the owner can access the data; and in a multiowner model, in addition to multiple queriers, the database has also several owners (MYKLETUN; NARASIMHA; TSUDIK, 2006). In this work, we consider a multiquerier model where all the data retrieved from a query belongs to the same owner. We also use the term "querier"or "client"as a single person or even a more general abstraction, such as an entity or company. Figure 7 depicts the considered model.

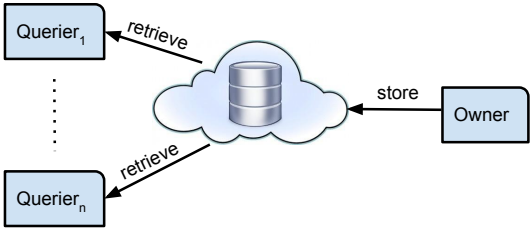

Figure 7: Multiquerier outsourced database model.

Although the data owners trust the maintenance of their data to an outsourced server, they (and all the clients who access their data) need mechanisms to guarantee the integrity and authenticity of these data, since they are stored in servers that can be untrusted or do not

process/store the data in a proper manner. Moreover, the data are accessed using an insecure communication channel and may be modified before reaching the querier. So we need to assure the tuples integrity in order to guarantee that they were not modified, and assure the tuples authenticity in order to authenticate the source of the data (in this case, their owners).

As presented in Mykletun et al. (MYKLETUN; NARASIMHA; TSUDIK, 2006), there are tree possible choices for the granularity of integrity: at the table level, tuple (or row) level and attribute (or column) level. The first refers to create one integrity check for the entire table, and in consequence this entire table need to be queried in order to verify even one single tuple integrity/authenticity, which is inefficient or even impractical. On the other extreme, if we create integrity checks at the level of attributes, it would result in a large overhead in terms of signature computation and verification for the owner and in terms of storage for the server. So, it is believed that a good compromise between this two extremes is to create integrity checks at the level of tuples.

In order to provide data integrity and authenticity, we need to choose an ideal solution to apply in a multiquerier model. One can see symmetric encryption schemes as an option for this application, since they are more efficient than the asymmetric ones. However, if we use any symmetric encryption scheme (such as MACs) in an outsourced database, only those who have access to the symmetric secret key, such as the owner of the data, are able to verify the integrity of the data. In multiquerier outsourced databases, other clients besides the owners must access the data and verify its integrity, so this would be a good solution for the unified client model, but not for the multiquerier model.

The use of asymmetric encryption (or more specifically, digital signatures) is ideal for a multiquerier outsourced database. In this scenario, the data owner has a key pair, he/she signs a tuple using the private key and the signature is stored with the tuple into the database. The public key is used by anyone who has access to this data in order to verify its signature. From now on, when we say that we are verifying the integrity we are automatically referring to authenticity too, since we use digital signature schemes and with a digital signature we can provide both.

Some computational aspects need to be taken into consideration to build a scheme of this nature. Mykletun et al. (MYKLETUN; NARASIMHA; TSUDIK, 2006) consider the following overhead sources that need to be minimized: the owner computation; the server storage and

computation; and the querier bandwidth and computation. Since the owner stores data with less frequency than queriers access them and the server storage nowadays is not a critical problem, in this work we focus on minimizing server computation, querier bandwidth, and querier computation.

Since we are considering the integrity check at the tuple level, we focus on SQL queries that give tuples as a result, not addressing the ones that return, for example, arithmetical results instead. Although the main focus of this work is to guarantee authenticity and integrity, some applications may require confidentiality and completeness, which are beyond the scope of this work. However, we assume that if the users want to keep their data private, they can store it encrypted on condition that it is still possible to query the data remotely. Moreover, Narasimha and Tsudik (NARASIMHA; TSUDIK, 2005) extend the work of Mykletun et al. (MYKLETUN; NARASIMHA; TSUDIK, 2006) in order to assure that the database returned all the possible tuples (completeness) by linking the tuples in order to create signature chains. The modification is only at the signature generation process to generate the chain, and some extra tuples are returned in order to verify it. We observe that it is possible to apply their method in our algorithms to additionally provide completeness.

In summary, we focus on verifying the integrity and authenticity of individually signed tuples returned from a multiquerier outsourced database. In this context, we aim to minimize the server computation, querier bandwidth, and querier computation. We note that the security of the method relies on the security of the signature algorithms used.

5.2 DEFINITION OF THE PROBLEM AND RELATED WORK

Symmetric and asymmetric encryption are used in several applications and some of them need to manage a large amount of data, as is the case in outsourced databases and sensor networks. Since the integrity verification process can be expensive, applications that have to verify a large number of integrity checks need methods to improve this process. There are methods that aim to send less data through the network and/or improve the verification steps. Some of them use cryptographic hash functions, such as the ones based on HMACs (KATZ; LINDELL, 2008; YAVUZ; NING, 2009), and others use asymmetric encryption (BONEH et al., 2003; FIAT, 1989; MYKLETUN; NARASIMHA; TSUDIK, 2006). As stated in Section 5.1, in this work we focus on using

asymmetric encryption (digital signatures) to provide authenticity and integrity checks.

Various methods in the literature propose the aggregation of digital signature as a solution to send less data through the network and improve the verification steps (BONEH et al., 2003; MYKLETUN; NARASIMHA; TSUDIK, 2006). Instead of sending n signatures with the n pieces of signed data, the methods combine these signatures into one without losing the ability to verify the integrity of the corresponding data. On the verification side, instead of verifying n signatures individually, only one verification is needed. It is possible to aggregate different types of signature, as long as they have the so-called homomorphism property (presented in Section 2.3).

In the context of outsourced databases, Mykletun et al. (MYKLETUN; NARASIMHA; TSUDIK, 2006) analyze the application of three well known signature schemes (BGLS (BONEH et al., 2003), DSA (KERRY; GALLAGHER, 2013) and RSA (RIVEST; SHAMIR; ADLEMAN, 1978)) and their aggregation. The BGLS aggregation scheme proposed by Boneh et al. (BONEH et al., 2003) is applicable to all the three outsourced database models (unified client, multiquerier, and multiowner). This method minimizes the bandwidth by aggregating all the signatures into one, but it can become too expensive on the verification side (MYKLETUN; NARASIMHA; TSUDIK, 2006). Additionally, early aggregation schemes for DSA signatures presented security issues (BOYD; PAVLOVSKI, 2000). Some more secure methods for aggregating DSA signatures were introduced, in which the verification costs decrease at the expense of sending more data through the network (BELLARE; GARAY; RABIN, 1998; NACCACHE et al., 1994). However, if we consider the multiquerier model where only one signer (and one private key) is involved, the aggregation of RSA signatures provides better results than BGLS in terms of querier computation, since the computation of bilinear maps is more expensive than modular exponentiation. In contrast with DSA aggregation schemes, the RSA aggregation presents better results for both querier computation and bandwidth.

In order to minimize the querier bandwidth, specially when the amount of signatures returned are considerably large, Mykletun et al. (MYKLETUN; NARASIMHA; TSUDIK, 2006) apply an aggregation of RSA signatures to be sent to the querier (condensed-RSA). Their method is also useful for decreasing the querier and server computation, since the querier needs to perform less cryptographic operations and the server only needs to aggregate the signatures (see the end of Section 2.3 for more details). However, if the querier receives even a single tuple

with an invalid signature, the integrity of the whole query is compromised. This is not desirable in situations where the integrity of a subset of a query result is still quite valuable information, specially in large query results, since some modified records invalidate the entire queried data. In order to not invalidate all the set of aggregated signatures it is necessary to precisely identify the faulty ones. This could only be done in all situations if the verifier of the signature has access to all the signatures individually, which is possible with batch verification but not with aggregated signatures (MYKLETUN; NARASIMHA; TSUDIK, 2006; ZAVERUCHA; STINSON, 2009).

In fact, we observe a *trade-off* between the two presented methods: by using aggregation it is possible to decrease the amount of data sent through the network, but only with batch verification we can identify all the faulty signatures. In this work, we consider a more flexible scheme that guarantees integrity and authenticity within the query results, identifying which entries have invalid signatures while transmitting a reasonable amount of data. We introduce the concept of *level of aggregation*, which assumes a maximum number of expected modified tuples and, consequently, the amount of aggregations the method needs to execute in order to identify them. We accomplish this by combining signature aggregation and combinatorial group testing, closely relating this work to (MYKLETUN; NARASIMHA; TSUDIK, 2006),(ZAVERUCHA; STINSON, 2009), and the MLSS method previously presented in Chapter 4. Here we define *Level-d Signature Aggregation Scheme* as a method, which produces a few aggregated signatures, that under the assumption of not more than d invalid signatures, allows the identification of all invalid signatures. This method is based on combinatorial group testing, which is discussed in Chapter 3.

One could also consider Merkle Hash Trees as a possible solution for decreasing the amount of signatures from a query result and identifying the invalid tuples. A Merkle Hash Tree is an authenticated data structure that is used to authenticate a set of values $x_1, \ldots, x_n$. The hash of these values are considered the tree leaves and the internal nodes are formed by its children's hash concatenation, and at the end the root of the tree is signed, guaranteeing the authenticity check of all the leaves. On the verification side we can verify $x_i{}'$ by using the concatenated internal hashes there are available, then combining it with the new $h(x_i{}')$ and verifying the root's signature (MERKLE, 1989). Devanbu et al. (DEVANBU et al., 2000) propose a solution for multiquerier outsourced databases using Merkle Hash Trees, but Mykletun et al. (MYKLETUN; NARASIMHA; TSUDIK, 2006) presents a few problems

in their solution for arbitrary querying. First, since only the root is signed, it would be necessary to precompute and store all the possible Hash Trees for all the possible queries, requiring a large storage and previous computation. Another problem comes with tuple updates, since each update will change the tuple hash and consequently all the structure of the trees that contains this tuple. So, solutions that do not involve precomputations and a large storage are ideal, such as the ones that involve aggregation and batch verification proposed in this work.

5.3 PROPOSED METHOD AND ALGORITHMS

Let us recall our main problem in multiquerier outsourced databases: when the server returns the results of a query (a set of n tuples) to the querier, it must also send an assurance of their integrity and authenticity. This assurance is based on the n signatures stored together with each of the n-tuples. If the priority of the outsourced database scheme is to have a small communication overhead and fast verification time, the server should send a single aggregated signature of the n original signatures. This is verified at the querier's end by a single aggregated signature verification. This is the scheme proposed by Mikletun et al. (MYKLETUN; NARASIMHA; TSUDIK, 2006). The drawback of this scheme is that a single invalid tuple invalidates all the query results. On the other hand, if the priority is to have a precise determination of the invalid tuples, then batch verification could be used at the verifier's end. However, this implies an increase in costs since it involves transmitting, together with the n tuples, n signatures in place of one aggregated signature as well as increasing the verification time at the querier's end. When the number n of tuples in the query result is large, the communication overhead and verification time may be prohibitive.

Our scheme addresses the trade-off between these two conflicting objectives. We propose a more flexible scheme in which the querier requests from the server that the query reply comes with level-d signature aggregation. We call level-0 aggregation the method that combines all signatures of the n tuples in the query result into one aggregated signature, done at the server's end, which corresponds to the scheme proposed in (MYKLETUN; NARASIMHA; TSUDIK, 2006). More generally, a level-d signature aggregation specifies a maximum number d of invalid tuples that the system support to be among the n tuples returned by the query, while still having a precise determination of the set of invalid

tuples. Using the CGT approach and a d-CFF(t, n) matrix as described in Chapter 3, a set of t-aggregated signatures is sent by the server and verified by the querier, where t is substantially smaller than n. In the other extreme, as d becomes large, we are required to have $t = n$ to achieve level-d signature aggregation. In this case, all n signatures are sent with the tuples and the querier must verify all of them, which can be done via a method of batch verification (ZAVERUCHA; STINSON, 2009). In this sense, our method can be seen as an intermediate solution between signature aggregation and batch verification, which is detailed in Section 5.5.

5.3.1 Algorithms

Before outlining the steps in our proposed scheme, it is important to describe the importance of a CFF matrix in the verification algorithm. When the matrix is a d-cover-free family, we simply use the tests that pass to determine the non-defective items, and Definition 1 in Chapter 3 guarantees that all non-defective items can be identified in this way as long as there are at most d defective ones. If this latter hypothesis is not true, the algorithm is able to detect this fact, and still returns valuable information, i.e. it returns a set that contains all the defective items and possibly a few more, which shows that the items outside this set are not defective. This classical CGT algorithm (see (DU; HWANG, 2000, Section 7.1)) based on CFF is detailed next, where it is adapted to the context of the current application. For this application, the testing algorithm for a set of tuples T is a verification of the aggregated signature σ^*, obtained by the chosen aggregation method, which we call AgreggateVerify(T, σ^*).

Algorithm CFFVerify $(M, d, (T_1, \ldots, T_n), (\sigma_1, \ldots, \sigma_t))$
  BadTuples $= \{1, 2, \ldots, n\}$
  for each $i = 1$ to t do
    $G_i = \{j : M_{i,j} = 1$ and $1 \leq j \leq n\}$;
    $TG_i = \{T_j : j \in G_i\}$;
    result = AgreggateVerify(TG_i, σ_i);
    if (result = true) then BadTuples = BadTuples $\backslash G_i$;
  end for
  if size(BadTuples) $\leq d$ then
      return (BadTuples, EXACT);
  else return (BadTuples, CONTAINEDHERE);

Now we present the aggregation and verification algorithms:

Client sends a query:
The querier sends a query Q and specifies the aggregation level d.

Server replies to the query:

1. The server computes the query Q obtaining the resulting n tuples $(T_1, \ldots, T_n)$, as well as their signatures $(\sigma_1, \ldots, \sigma_n)$.
2. The server employs a deterministic algorithm; let us call it ComputeCFF(n, d), that returns a d-CFF(t, n) matrix M using the methods mentioned in Chapter 3.
3. The server performs t different signature aggregations of different subsets of $\{\sigma_1, \ldots, \sigma_n\}$, according to the rows of matrix M, producing t aggregated signatures $(\sigma_1^*, \ldots, \sigma_t^*)$.
4. The server sends $(T_1, \ldots, T_n)$ and $(\sigma_1^*, \ldots, \sigma_t^*)$ to the client.

Client receives the query results with level-*d* signature aggregation:

1. The querier receives $(T_1, \ldots, T_n)$ and $(\sigma_1^*, \ldots, \sigma_t^*)$.
2. The querier uses the same deterministic algorithm as the server, ComputeCFF(n, d), that returns the d-CFF(t, n) matrix M.
3. The querier runs the verification algorithm:
   CFFVerify$(M, d, (T_1, \ldots, T_n), (\sigma_1^*, \ldots, \sigma_t^*))$,
   which returns (BadTuples, Result).
4. The querier computes the set of tuples that are guaranteed to be valid:
   GoodTuples $= \{1, 2, \ldots, n\} \backslash$ BadTuples.
5. If Result=EXACT then the set GoodTuples is complete. Otherwise, the querier may decide to increase d to a value at most #(BadTuples) and restart the query process.

We remark that the algorithm used for signature aggregation and verification (Step 3 of both server and client methods) is general, and could use any of the applicable methods of aggregation described in Section 2.3. In the next section, we give a more detailed analysis of the performance of this approach when we aggregate signatures using condensed-RSA, comparing with the approach in (MYKLETUN; NARASIMHA; TSUDIK, 2006).

5.4 ANALYSIS

In Table 10, we give some concrete data to exemplify the quantity t, that is, the number of aggregated signatures for a level-d scheme for a query returning n tuples. For $d = 1$ we use the known optimal method that minimizes t, while for $d \geq 2$, we employ the methods that give the smaller t, based in the values presented in Table 4 of Chapter 3. For instance, if a query returns 1 million tuples it needs to provide only 23 aggregated signatures for level-1 and 346 for level-4.

Table 10: Number t of aggregations for given d, n.

d/n	$\mathbf{10^3}$	$\mathbf{10^4}$	$\mathbf{10^5}$	$\mathbf{10^6}$
1	13	16	20	23
2	63	83	104	125
3	127	148	185	222
4	159	231	288	346
5	190	332	415	498
10	348	1100	1394	1672

Here we analyze the application of the general method presented in Section 5.3 to the case of condensed-RSA signature scheme. Each of the t aggregations on the server side is given by the multiplication of the corresponding tuple signatures. The querier performs t condensed-RSA verification algorithms, each one consisting of multiplying the hashes of a specified subset of tuples T_i and comparing the result with the decrypted version of σ_i.

Let $|\sigma|$ be the size of an RSA signature, *mult* be the cost of one modular multiplication and *exp* be the cost of one modular exponentiation. The querier bandwidth requirement is $t \times |\sigma|$. For level-d aggregation we use a CFF matrix M with t rows and w ones as discussed in Chapter 3. On the server side, the aggregation consists of multiplying the signatures based on M, which means t rounds of signature multiplications are done involving a total of w signatures. Thus, the server computation costs $(w - t)$ *mult*. On the verification end, the querier needs to decrypt t aggregated signatures, generating a cost of t exponentiations as well as performing the same aggregations as the server to compute the tuples' integrity, which consists on $w - t$ multiplications. Thus, the querier computation consists of t *exp* $+(w - t)$ *mult*. For level-n we have no restrictions on the maximum number of invalid signatures d that we can identify, but it requires all the n signatures

to be sent individually (as in batch verification). At the verification end it is necessary to combine the n signatures in order to improve the verification, which gives us $n-1$ multiplications of signatures, $n-1$ multiplications of hashes and one exponentiation (as seen in (HARN, 1998b)).

Table 11 presents the costs of level-d aggregated RSA signatures, condensed-RSA, and level-n aggregated RSA signatures, comparing querier bandwidth (QB), server computation (SC), querier computation (QC), and ability to identify invalid signatures (Identify). The costs for level-d aggregated RSA depends on t and w. The costs for level-n only consider the verification of the signatures. If there is any invalid signature, it is necessary to apply methods as the ones seen in Section 2.5.

Table 11: Comparisons between condensed-RSA, level-d and level-n signature aggregation.

	QB	**SC**	**QC**	**Identify**
Condensed-RSA	$\|\sigma\|$	$(n-1)$ *mult*	$(n-1)$ *mult* + 1 *exp*	NO
Level-d sig. agg.	$t \times \|\sigma\|$	$(w-t)$ *mult*	$(w-t)$ *mult* + t *exp*	YES up to d
Level-n sig. agg.	$n \times \|\sigma\|$	0	$(2(n-1))$ *mult* + 1 *exp*	YES

To analyze these quantities more precisely, we would like to write t and w in terms of d and n. As seen in Chapter 3, when $d = 1$ we have $t \sim \log_2 n$ and $w = (t/2) \times n$. For the case $d \geq 2$, various constructions are possible, but we consider the construction that takes over as n grows, yielding $t = (d+1)^2 \ln n$ and $w = \frac{(d+1)}{2} n \ln n$. Based on these values, we produce Table 12, which reveals that our method is more expensive and by how much. For level-1 aggregation, the query bandwidth and server computations have their costs multiplied by a factor of about $\log n$. For level-d aggregation, the bandwidth is multiplied by $(d+1)^2 \ln n$ and the server cost is multiplied by $\frac{(d+1)}{2} \ln n$; the querier cost has an extra factor of $\frac{(d+1)}{2} \ln n$ for multiplications and of $(d+1)^2 \ln n$ for exponentiations. Our extra costs are incurred but give us the ability of identification of invalid signatures.

5.5 TRADEOFF BETWEEN AGGREGATION AND BATCH VERIFICATION

Aggregation of signatures provide a small overhead by aggregating all the signatures into one before sending them to the verifier.

Table 12: Comparison between condensed-RSA and level-d signature aggregation.

	QB	**SC**	**QC**
Cond.	$\|\sigma\|$	$(n-1)$ *mult*	$(n-1)$ *mult* + 1 *exp*
Level-1	$\log_2 n \times \|\sigma\|$	$\frac{(n-2)\log_2 n}{2}$ *mult*	$\frac{(n-2)\log_2 n}{2}$ *mult* + $\log_2 n$ *exp*
Level-d	$(d+1)^2 \ln n \times \|\sigma\|$	$(d+1)(\frac{n}{2} - d - 1)\ln n$ *mult*	$(d+1)(\frac{n}{2} - d - 1)\ln n$ *mult* + $(d+1)^2 \ln n$ *exp*
Level-n	$n \times \|\sigma\|$	0	$(2(n-1))$ *mult* + 1 *exp*

On the verification side, if all the signatures are valid, the verification succeed, but if some signatures are invalid the verifier can not identify them. This is due to the fact that after aggregating all the signatures together, we can not "disintegrate"them in order to have access to the signatures individually and identify the invalid ones (MYKLETUN; NARASIMHA; TSUDIK, 2006; ZAVERUCHA; STINSON, 2009).

When all the signatures are sent individually with their respective data, batch verification speeds up the verification process by checking all the signatures in one step. This is important specially in the cases where the verification costs several operations, as modular exponentiations in the case of RSA (MALINA; HAJNY; ZEMAN, 2013). If all the signatures are valid the batch verification succeed, but if some of them are invalid the verification fails and it is necessary to identify them. By sending each signature separately, batch verification has the opportunity to locate the invalid signatures (ZAVERUCHA; STINSON, 2009), but it results in a large communication overhead since all the signatures are sent individually, without any compression.

Consider a general application where several data are signed (messages, documents, tuples, files, etc.) and need to be sent through the network and verified at the other side. The level-d aggregation method proposed in Section 5.3 can solve this trade-off between aggregation and batch verification. In this proposed method we send a small amount of signatures (t aggregated signatures) while allowing the exact location of invalid signatures if their number is at most d, contrasting with batch that can achieve this by sending all the n signatures and with aggregation that sends only one signature but can not locate the invalid ones. This is only possible in our method because we aggregate them using combinatorial group testing constructions, which allows the identification of invalid signatures with a few tests (t aggregated signature verifications). If there are no limitations on bandwidth, our method can also support batch verification by constructing a level-n

signature aggregation, where we consider $d = n$ and send all the signatures individually. In this case it is possible to locate all the invalid signatures by applying a localization method (as the ones seen in Section 2.5), since $d = n$. The comparison of the methods is illustrated in Figure 8. More details on this comparison, with the number of operations for specific and general values of d, are given in Tables 11 and 12.

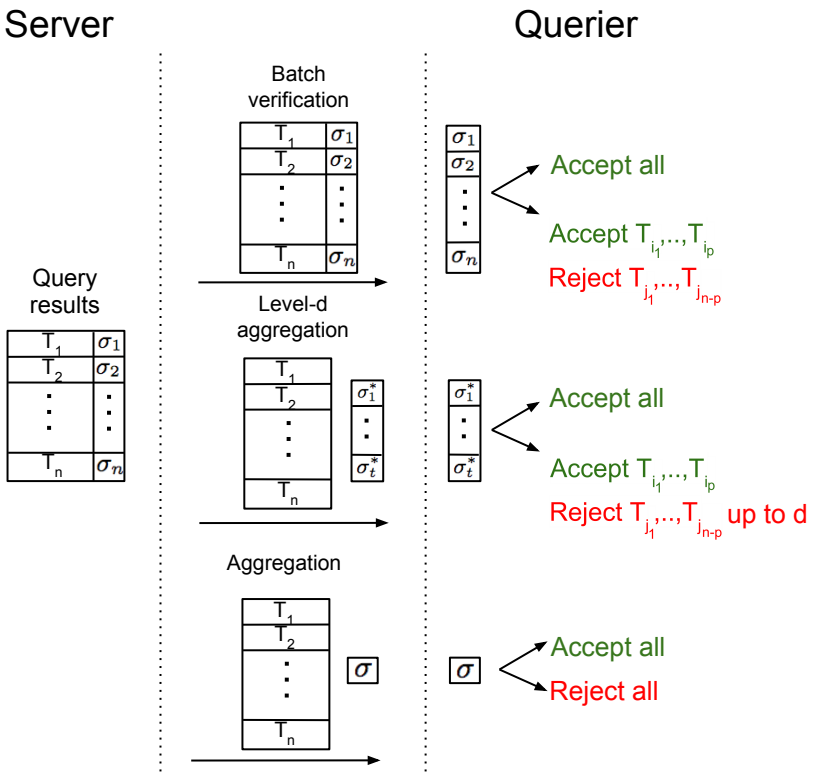

Figure 8: Comparison between methods.

5.6 FINAL CONSIDERATIONS

In this chapter we consider the problem of ensuring the integrity and authenticity of tuples outsourced in a multiquerier database. We allow the querier to select levels of aggregation, being able to identify the modified tuples if the signatures do not match. This scheme can be considered a trade-off between the methods of aggregation of signatures and batch verification, since it provides a way to send a small amount of signatures through the network without losing the ability to identify the invalid tuples.

We remark that our method can be applied in conjunction with any signature scheme that has the homomorphic property, and not only condensed-RSA. Our method is also applicable for multiowner outsourced databases, by employing appropriate aggregation methods that work for that scenario, and clearly for the unified client model.

The ideas presented here are also useful beyond the database scenario, as the proposed method can be used in any application that needs to aggregate and verify digital signatures. When the communication bandwidth is a concern, this method provides a trade-off between signature aggregation and batch verification. Indeed, our approach differs from the batch verification via group testing discussed in (ZAVERUCHA; STINSON, 2009), in that more work is required from the sender in order to reduce the amount of signatures sent to the verifier, while in batch verification all signatures are sent.

6 FINAL CONSIDERATIONS

The contribution of this work is the proposal of signature schemes with additional information capable of locating modifications for the cases where the signature verification does not match. We propose two different schemes that can partially guaranty the integrity and authenticity of signed data or a set of individually signed data.

With traditional digital signature schemes we can only have a boolean answer when checking a document's signature: the signature matches the document or it does not. In our first scheme we propose the Modification Location Signature Scheme (MLSS), which builds the document signature with some additional information, and it can be used to locate the exact portion of the document that was modified. In this scheme, we divide the document into blocks and use nonadaptive combinatorial group testing to construct its signature. At the verification process, if the signature does not match, we can use the information generated using group testing to exactly locate the modified blocks in an efficient way.

With MLSS we can allow modifications in some expected portions of signed documents (useful for fillable forms), we can use the modified parts for investigation (forensics), or even guarantee the integrity of the remaining portions and use the information contained there (not modified tuples in a database). However, if the verifier is not interested in locating the modifications, the algorithm was designed to respect this choice and verify the signature in a way similar to traditional algorithms, ignoring the extra information.

We can also see the need of partial integrity in scenarios where we have several data individually signed that need to be sent and verified. In order to improve the verification process and save the bandwidth overhead, we can use mechanisms such as aggregation of signatures and batch verification. With aggregation we can combine all the n signatures into one, saving bandwidth, and we can also speed up the verification process. However, if the verification does not match we need to invalidate the entire set of data/signatures, since we do not have enough information to identify the invalid signatures. With batch verification we send all the data/signatures individually and combine them at the verifier's end to speed up the verification process. Here we do not save bandwidth, but if the verification does not match, we still have access to all the signatures individually in order to identify the invalid ones.

Taking into consideration the trade-off between these two methods, we propose a level-d signature aggregation scheme able to compact the digital signatures into a reasonable size while allowing the identification of the invalid signatures at the verification process. We use nonadaptive combinatorial group testing algorithms to generate t aggregated signatures (with $t \sim O(\log n)$), and we send these t signatures with the data (instead of sending n signatures with batch verification). At the other side, we verify the t aggregated signatures and if any of them do not match, since the aggregation was built using group testing, we can exactly locate the signature(s) responsible for the failure. With this scheme we have an intermediate solution between aggregation and batch verification, since we decrease the amount of signatures sent through network without losing the ability to identify the invalid ones.

We consider this scheme in a real scenario of outsourced databases, where each tuple is individually signed and stored in an untrusted server. We apply the level-d aggregation on the tuples that are returned from a query by partially aggregating the tuples signatures. So, we return less signatures to the verifier and we are still able to localize the modified tuples if the verification fails. We consider signatures generated using the RSA algorithm and compare our method with traditional schemes. However, our method can be used with any signature algorithm that supports aggregation or batch, as the ones presented in related works, and not only with RSA.

Both methods are based on nonadaptive combinatorial group testing, in particular they are designed using matrix constructions that present the cover-free property. We present detailed discussion about the best algorithms that can be used to construct these matrixes, and recommend specific constructions for different parameters used.

In terms of future work, we suggest a few extensions to the MLSS scheme. The first improvement is to design an specific cover-free matrix to be used in this application, which not only keep t small but also decreases w, in order to improve the costs with hash calculations. Another interesting future work is to apply this method to a real scenario of digital documents, such as implementing an application to sign and verify PDF documents using MLSS.

We also suggest some interesting future work in the level-d aggregation scheme. The first suggestion is to consider the scheme in other important applications, such as sensor networks. Moreover, it is important to consider not only RSA, but also different signature algorithms and their applications. Finally, we consider only signatures

generated by the same key, an extension of the method to consider signatures from different signers would also be interesting.

BIBLIOGRAPHY

BARRETO, P. S. L. M. **Criptografia robusta e marcas d'água frágeis: construção e análise de algoritmos para localizar alterações em imagens digitais.** Tese (Doutorado) — Universidade de São Paulo, 2003.

BELLARE, M.; GARAY, J. A.; RABIN, T. Fast batch verification for modular exponentiation and digital signatures. In: . [S.l.]: Springer-Verlag, 1998. p. 236–250.

BELLARE, M.; ROGAWAY, P. Randon oracles are practical: a paradigm for designing efficient protocols. p. 62–73, 1993.

BONEH, D. et al. Aggregate and verifiably encrypted signatures from bilinear maps. In: **Proceedings of the 22nd international conference on Theory and applications of cryptographic techniques**. Berlin, Heidelberg: Springer-Verlag, 2003. (EUROCRYPT'03), p. 416–432. ISBN 3-540-14039-5. Disponível em: <http://dl.acm.org/citation.cfm?id=1766171.1766207>.

BONEH, D.; LYNN, B.; SHACHAM, H. Short signatures from the weil pairing. p. 514–532, 2001.

BONIS, A. D.; CRESCENZO, G. D. Combinatorial group testing for corruption localizing hashing. In: **COCOON**. [S.l.]: Springer, 2011. v. 6842, p. 579–591.

BONIS, A. D.; CRESCENZO, G. D. A group testing approach to improved corruption localizing hashing. In: . [S.l.: s.n.], 2011. v. 2011, p. 562.

BOYD, C.; PAVLOVSKI, C. Attacking and repairing batch verification schemes. p. 58–71, 2000.

CAMENISCH, J.; HOHENBERGER, S.; PEDERSEN, M. Ø. Batch verification of short signatures. Springer Verlag, 2011.

CRESCENZO, G. D.; JIANG, S.; SAFAVI-NAINI, R. Corruption-localizing hashing. In: **ESORICS**. [S.l.]: Springer, 2009. p. 489–504.

DEVANBU, P. T. et al. Authentic third-party data publication. In: . [S.l.]: Kluwer, 2000. v. 201, p. 101–112.

DIFFIE, W.; HELLMAN, M. E. New directions in cryptography. **IEEE Transactions on Information Theory**, v. 22, n. 6, p. 644–654, 1976.

DU, D.-Z.; HWANG, F. K. **Combinatorial group testing and its applications**. [S.l.]: World Scientific, 2000.

ELGAMAL, T. A public key cryptosystem and a signature scheme based on discrete logarithms. In: **Advances in Cryptology**. [S.l.: s.n.], 1985, (Lecture Notes in Computer Science). p. 10–18.

ERDöS, P.; FRANKL, P.; FUREDI, Z. Families of finite sets in which no set is covered by the union of r others. **Israel Journal of Mathematics**, v. 51, p. 79–89, 1985.

FERRARA, A. L. et al. Practical short signature batch verification. In: **Proceedings of the The Cryptographers' Track at the RSA Conference 2009 on Topics in Cryptology**. Berlin, Heidelberg: [s.n.], 2009. p. 309–324.

FIAT, A. Batch RSA. In: **Proceedings on Advances in Cryptology**. New York, NY, USA: Springer-Verlag New York, Inc., 1989. (CRYPTO '89), p. 175–185. ISBN 0-387-97317-6.

GOODRICH, M. T.; ATALLAH, M. J.; TAMASSIA, R. Indexing information for data forensics. In: **ACNS**. [S.l.: s.n.], 2005. (Lecture Notes in Computer Science), p. 206–221.

HARN, L. Batch verifying multiple dsa-type digital signatures. **Electronics Letters**, v. 34, n. 9, p. 870–871, 1998.

HARN, L. Batch verifying multiple rsa digital signatures. **Electronics Letters**, v. 34, n. 12, p. 1219–1220, 1998.

IDALINO, T. B. et al. Locating modifications in signed data for partial data integrity. **Information Processing Letters**, 2015. Disponível em: <http://dx.doi.org/10.1016/j.ipl.2015.02.014>.

JOHNSON, D.; MENEZES, A. **The elliptic curve digital signature algorithm (ECDSA)**. [S.l.], 1999.

JOHNSON, R. et al. Homomorphic signature schemes. **CT-RSA 2002 LNCS**, p. 244–262, 2002.

KATZ, J.; LINDELL, Y. Aggregate message authentication codes. In: **Proceedings of CT-RSA, LNCS 4964**. [S.l.: s.n.], 2008.

KAUTZ, W.; SINGLETON, R. Nonrandom binary superimposed codes. **Information Theory, IEEE Transactions on**, v. 10, p. 363–377, 1964. Disponível em: <http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1053689>.

KERRY, C. F.; GALLAGHER, P. D. **FIPS PUB 186-4 FEDERAL INFORMATION PROCESSING STANDARDS PUBLICATION Digital Signature Standard (DSS)**. [S.l.], 2013.

LAW, L.; MATT, B. J. Finding invalid signatures in pairing-based batches. In: . [S.l.: s.n.], 2007. (Lecture Notes in Computer Science), p. 34–53. ISBN 978-3-540-77271-2.

LU, S. et al. Sequential aggregate signatures, multisignatures, and verifiably encrypted signatures without random oracles. In: . [S.l.: s.n.], 2012.

LYSYANSKAYA, A. et al. Sequential aggregate signatures from trapdoor permutations. In: **Advances in Cryptology, EUROCRYPT 2004**. [S.l.]: Springer-Verlag, 2004. p. 74–90.

A.T. Lytle, B.J. Reich, G. Gupta, M.C. Pohle e M. Tikunova. **Techniques for digital signature formation and verification**. maio 29 2012. US Patent 8,190,902. Disponível em: <https://www.google.com/patents/US8190902>.

MALINA, L.; HAJNY, J.; ZEMAN, V. Trade-off between signature aggregation and batch verification. In: **36th International Conference on Telecommunications and Signal Processing (TSP), 2013**. [S.l.: s.n.], 2013. p. 57 – 61. ISBN 978-1-4799-0402-0.

MARTIN, K. **Everyday Cryptography**. [S.l.]: Oxford University Press, 2012. ISBN 978-0-19-969559-1.

MATT, B. J. Identification of multiple invalid signatures in pairing-based batched signatures. In: **Proceedings of the 12th International Conference on Practice and Theory in Public Key Cryptography: PKC '09**. Berlin, Heidelberg: Springer-Verlag, 2009. p. 337–356.

MENEZES, A. J.; VANSTONE, S. A.; OORSCHOT, P. C. V. **Handbook of Applied Cryptography**. 1st. ed. Boca Raton, FL, USA: CRC Press, Inc., 1996. ISBN 0849385237.

MERKLE, R. C. A certified digital signature. In: **Proceedings on Advances in Cryptology**. [S.l.]: Springer-Verlag New York, Inc., 1989. p. 218–238.

MYKLETUN, E.; NARASIMHA, M.; TSUDIK, G. Authentication and integrity in outsourced databases. **Trans. Storage**, ACM, New York, NY, USA, v. 2, n. 2, p. 107–138, maio 2006. ISSN 1553-3077. Disponível em: <http://doi.acm.org/10.1145/1149976.1149977>.

NACCACHE, D. et al. Can dsa be improved? – complexity trade-offs with the digital signature standard. p. 85–94, 1994.

NARASIMHA, M.; TSUDIK, G. Dsac: Integrity for outsourced databases with signature aggregation and chaining. **ACM Conference on Information and Knowledge Management**, p. 235–236, 2005.

NEVEN, G. Efficient sequential aggregate signed data. In: **EUROCRYPT**. [S.l.: s.n.], 2008. p. 52–69.

PASTUSZAK, J. et al. Identification of bad signatures in batches. In: **Public Key Cryptography**. [S.l.]: Springer, 2000. (Lecture Notes in Computer Science, v. 1751), p. 28–45. ISBN 3-540-66967-1.

PASTUSZAK, J.; PIEPRZYK, J.; SEBERRY, J. Codes identifying bad signature in batches. In: **INDOCRYPT**. [S.l.]: Springer, 2000. (Lecture Notes in Computer Science), p. 143–154.

PORAT, E.; ROTHSCHILD, A. Explicit nonadaptive combinatorial group testing schemes. **IEEE Transactions on Information Theory**, p. 7982–7989, 2011.

PRENEEL, B. The first 30 years of cryptographic hash functions and the NIST SHA-3 competition. In: **Proceedings of the 2010 international conference on Topics in Cryptology**. Berlin, Heidelberg: Springer-Verlag, 2010. (CT-RSA'10), p. 1–14. ISBN 3-642-11924-7, 978-3-642-11924-8. Disponível em: <http://dx.doi.org/10.1007/978-3-642-11925-5_1>.

RIVEST, R.; ADLEMAN, L.; DERTOUZOS, M. On data banks and privacy homomorphisms. **Foundations of Secure Computation**, p. 169–177, 1978.

RIVEST, R. L.; SHAMIR, A.; ADLEMAN, L. M. A method for obtaining digital signatures and public-key cryptosystems. **Commun. ACM**, v. 21, n. 2, p. 120–126, 1978.

SCHNEIER, B. **Applied Cryptography**. [S.l.: s.n.], 1996.

SPERNER, E. Ein Satz über Untermengen einer endlichen Menge. p. 544–548, 1928.

STEINFELD, R.; BULL, L.; ZHENG, Y. Content extraction signatures. p. 285–304, 2002.

STINSON, D. R.; WEI, R. Generalized cover-free families. **Discrete Math**, v. 279, p. 463–477, 2002.

WEGMAN, M. N.; CARTER, J. New hash functions and their use in authentication and set equality. **Journal of Computer and System Sciences**, v. 22, n. 3, p. 265 – 279, 1981.

YAVUZ, A. A.; NING, P. Hash-based sequential aggregate and forward secure signature for unattended wireless sensor networks. IEEE, Toronto, ON, p. 1–10, 2009.

ZAVERUCHA, G. M.; STINSON, D. R. Group testing and batch verification. In: KUROSAWA, K. (Ed.). **ICITS**. [S.l.]: Springer, 2009. (Lecture Notes in Computer Science, v. 5973), p. 140–157.