Modification tolerant signature schemes: location and correction

Thais Bardini Idalino*, Lucia Moura, and Carlisle Adams

University of Ottawa, Ottawa, Canada
{tbardini,lmoura,cadams}@uottawa.ca

October 2019

Abstract. This paper considers malleable digital signatures, for situations where data is modified after it is signed. They can be used in applications where either the data can be modified (collaborative work), or the data must be modified (redactable and content extraction signatures) or we need to know which parts of the data have been modified (data forensics). A classical digital signature is valid for a message only if the signature is authentic and not even one bit of the message has been modified. We propose a general framework of modification tolerant signature schemes (MTSS), which can provide either location only or both location and correction, for modifications in a signed message divided into $n$ blocks. This general scheme uses a set of allowed modifications that must be specified. We present an instantiation of MTSS with a tolerance level of $d$, indicating modifications can appear in any set of up to $d$ message blocks. This tolerance level $d$ is needed in practice for parametrizing and controlling the growth of the signature size with respect to the number $n$ of blocks; using combinatorial group testing (CGT) the signature has size $O(d^2 \log n)$ which is close to the best known lower bound of $\Omega(\frac{d^2}{\log d}(\log n))$. There has been work in this very same direction using CGT by Goodrich et al. (ACNS 2005) and Idalino et al. (IPL 2015). Our work differs from theirs in that in one scheme we extend these ideas to include corrections of modification with provable security, and in another variation of the scheme we go in the opposite direction and guarantee privacy for redactable signatures, in this case preventing any leakage of redacted information.

Keywords: Modification tolerant signature; redactable signature; content extraction signature; malleable signature; modification localization; modification correction; digital signature; combinatorial group testing; cover-free family.

1 Introduction

Classical digital signature schemes (CDSS) are used to guarantee that a document was created by the sender (authenticity) and has not been modified along

* Funding granted from CNPQ-Brazil [233697/2014-4] and OGS.
the way (integrity); they also help to prevent the signer from claiming s/he did not sign a message (non-repudiation). The verification algorithm has a boolean output: a successful outcome is achieved if and only if both the signature is valid and the document has not been modified. In this paper, we consider more general digital signature schemes which we call modification-tolerant signature schemes (MTSS), which go beyond the ability of detecting modifications provided by CDSS, and have the ability of locating modifications or locating and correcting modifications. We discuss two types of modification-tolerant signature schemes: a general MTSS that allows the location of modified blocks of the data, and an MTSS with correction capability, that allows the correction of the modified blocks, recovering the original message. We give three instantiations of the scheme for the purpose of location, correction, and redaction.

In which situations can modifications be allowed or even desired in the context of digital signatures? One situation where modifications are desirable is the so-called redactable signatures [13], also called content extraction signatures [21]. In redactable signature schemes [13,18,21], some blocks of a document are redacted (blacked out) for privacy purposes, without interacting with the original signer and without compromising the validity of the signature. Related to these are the “sanitizable” signatures introduced by Ateniese et al. [1] which are also used for the purpose of privacy of (parts of) the original message, but the modifications are done by a semi-trusted third party (the sanitizer) who can modify blocks of the document and sign the modified version, without the need of intervention by the original signer. Thus, in both redactable and sanitizable signature schemes, the privacy of the (redacted or modified parts of the) original message must be preserved. For MTSS, privacy of the original data that has been modified is not required. An MTSS with only location capability that guarantees privacy can be used for implementing redactable signatures, but that is not an intrinsic requirement for MTSS. Indeed, as pointed out in [18] the scheme provided in [12] does not meet standard privacy requirements. In the case of MTSS with correction capability, privacy cannot be guaranteed by definition, since the method permits the recovery of the original document.

A different scenario where a moderate amount of modification is desirable involves collaborative work. The authors of a document can use MTSS to allow further modifications as long as the original parts can be identified as their own. Other collaborators may apply modifications to the original document and append the original document’s signature, which provides information about which blocks were modified, as well as a guarantee of integrity of the original blocks. MTSS can separate the original blocks from the modified ones, while correction capability further provides retrieval of the original version of the document.

Locating modifications has also been considered in situations where modifications are not desired, but their localization is used as a mechanism to mitigate damage. Indeed, in the context of message authentication codes for data structures, Goodrich et al. [11] propose a message authentication code (MAC) with modification locating capabilities. They propose their use in data forensics applications since the identification of which information was modified can be used
to identify the perpetrator of the crime (for example: the salary of a person or the grade of a student was modified on a database). In [12], in the context of MTSS with only location capability, the authors mention the advantage of being able to guarantee the integrity of part of the data instead of the all-or-nothing situation given by a CDSS boolean outcome. For example, the integrity of 95% of a document or database may contain enough information needed for a specific application, whereas it would have been considered completely corrupted and unusable in the case of CDSS. In the case of MTSS with correction capability, we can go beyond mitigating the damage, and actually recover the originally signed document.

The mechanism behind the MTSS schemes instantiated here, like in [11,12], is the use of cover-free families (CFF) in the same way as it is traditionally employed in combinatorial group testing. Combinatorial group testing has been used in cryptography in the context of digital signatures [23], broadcast communication [8,20], and many others. The main idea is to test \( t \) groups, which are subsets of the \( n \) blocks, together (with enough redundancy and intersections between groups), and each group is used to produce a distinct hash. The tuple of \( t \) hashes is then signed and provided that no more than \( d \) blocks were modified, it is possible to identify precisely which blocks have been modified. The main efficiency concern is the compression ratio: the order of growth of \( n/t \) as \( n \) grows, for a fixed modification tolerance \( d \). Using cover-free families, it is possible to achieve a compression ratio of \( O(n/(d^2 \log n)) \), which is not much worse than the \( O(n) \) compression ratio given by modification intolerant schemes such as CDSS.

**Our contributions:** In the present paper, we propose a general framework for MTSS, and a specific MTSS scheme for modification correction and another for redacted signatures. Both schemes are based on a modification tolerance \( d \), indicating the maximum number of modifications or redactions. The security of the schemes rely only on an underlying classical digital signature scheme and on collision resistant hash functions; therefore the schemes can be used in the context of postquantum cryptography as long as these underlying building blocks are postquantum secure.

First, we extend methods that use cover-free families for modification location [11,12] to further provide modification correction (Scheme 2 in Section 4.2), provided that the size of the blocks are small enough, say bounded by a constant \( s \) where an algorithm with \( 2^s \) steps can run in an acceptable amount of time. In short, the localization of the modified blocks is done using CFF similarly to [11,12] with an extra constraint on blocks of size at most \( s \), and an exhaustive search is used to correct the block to a value that makes the concatenation of a specific group of blocks match its original hash. The assumption that a collision resistant hash function does not cause collisions for messages of small size up to \( s \) is not only reasonable, but can be tested before employing it in the method.

Second, we propose a variation of the scheme for modification location to ensure total privacy of the modified blocks in order to extend it for the purpose of redactable signatures (Scheme 3 in Section 6). In this case, a block modification...
can be a redaction to hide private information. Unlike the modification correction scheme, this scheme does not need a restriction on block size.

**Paper Organization:** In Section 2, we discuss related work. In Section 3, we give a general framework for modification tolerant signature schemes with and without modification correction. In Section 4, we instantiate the schemes for location and/or correction that allows any modifications in up to \(d\) blocks using cover-free families. In Section 5, we prove the unforgeability of the schemes under the adaptive chosen message attack. In Section 6, we extend and instantiate the modification location scheme for redactable signatures, and prove it guarantees total privacy. In Section 7, we discuss implementation issues of the schemes proposed in this paper. In Section 8, we consider the relationship of parameters and the impact on the size of the signature and the ability to locate and correct modifications. A conclusion is given in Section 9.

## 2 Related Work

The idea of error detection and correction of corrupted data is well established in coding theory. Combinatorial group testing can help locating where errors (modification of data) occurred, which can be considered an intermediate goal, which is stronger than error detection and more efficient than error correction. In the context of cryptography, localization of modified portions of data has been studied in the context of hash functions [4,5], digital signatures [3,12], and message authentication codes [7,11]. Correction of modifications is proposed by some schemes, but they have severe limitations on the correction capability [3,7]. Schemes such as the ones in [4,5,11,12] use techniques related to cover-free families to generate redundant information, which is used later to locate the modified portions of the data. The benefit of cover-free families is that they require a small amount of redundant data for a fixed threshold \(d\) in location capability.

The methods that provide location at the hash level [4,5] are based on superimposed codes, which are binary codes equivalent to \(d\)-cover-free families (\(d\)-CFF). These codes are used in two steps of the process: to generate a small set of hash tags from an input message, and to verify the integrity of the message using these tags. Because of the \(d\)-CFF property, the scheme allows the identification of up to \(d\) corrupted data segments [4,5]. A digital signature scheme with location of modifications is presented in [12]. Their scheme uses \(d\)-CFF matrices to generate a digital signature scheme that carries extra information for location of modifications. In this case, data is divided into \(n\) blocks which are concatenated and hashed according to the rows of the \(d\)-CFF. This set of hash values is signed using any classical digital signature algorithm, and both the signature and the set of hashes form the new signature of the message. The verification algorithm uses this information to precisely locate up to \(d\) blocks that contain modified content [12]. This is presented in detail in Section 4.2 as Scheme 1. In [11], the authors propose a solution for locating the items that were modified inside a data structure. They compute a small set of message authentication tags based on the rows of a \(d\)-disjunct matrix, which is equivalent to \(d\)-CFFs. These
tags are stored within the topology of the data structure, and an auditor can later use these tags to determine exactly which pieces of data were modified and perform forensics investigation.

The methods described above have a common approach: they all compute extra integrity information based on combinatorial group testing techniques and use this information to locate up to $d$ modified pieces of data. In our work, we show that if these pieces are small enough, it is possible to actually correct them to the originally-signed values. In the literature, we find signature and message authentication schemes that provide the correction of modifications, such as in [3,7]. These schemes use different approaches than the ones previously discussed, and the capacity of location and correction is very limited. In [3] only a single error can be corrected, while in [7] the data is divided into blocks, and only one bit per block can be corrected.

Error correcting codes ($d$-ECC) are related to this work, as we could use them to provide authentication and correction of $d$ modifications as we explain next. For a message $m$ formed by blocks of up to $\log_2 q$ bits each, sign it using a CDSS and compute the corresponding codeword $c_m$ using a $d$-ECC over alphabet $q$. Send the codeword $c_m$ and the signature $\sigma$. The receiver obtains $c'$, which may differ from $c_m$ in at most $d$ blocks, and by decoding $c'$, obtains $c_m$ and thus the original message $m$. Then, the verifier can check its authenticity and integrity by verifying $(m, \sigma)$. We call this scheme Scheme E. In Section 4.3 we compare Scheme E and Scheme 2 using $d$-CFF, and find that they have very similar compression ratios. Scheme 2 has the advantage of giving more information in the failing case, of being more efficient when we do not need correction, and of being a simple variation of Schemes 1 and 3, facilitating comparison among the different approaches.

There is also a solution proposed in [2] for location of modifications in images with digital signatures, using a different approach than $d$-CFFs. Their scheme consists of partitioning the image into $n$ blocks, and generating one digital signature per block (with dependency on neighbours to avoid attacks). Although we can use signature schemes with very small signature sizes, the total number of signatures in this scheme is linear with the number of blocks $n$, while in [12], for example, the amount of extra information produced is logarithmic in $n$ because of the $d$-CFF.

Redactable or content extraction digital signatures [6,13,21] are used when the owner of a document signed by another party, needs to show parts of that document to a third party, but wants to keep parts of the document private. In Section 6 we give a variation of an MTSS scheme that implements redactable signatures. More on related work involving redactable signatures is discussed in that section.
3 Framework for modification-tolerant digital signatures

3.1 Classical Digital Signature Schemes (CDSS)

Classical digital signature schemes are based on public-key cryptography and consist of three algorithms: KEYGENERATION, SIGN, and VERIFY. We consider that any document or piece of data to be signed is a sequence of bits called a message.

**Definition 1.** A classical digital signature scheme (CDSS) is a tuple \( \Sigma \) of three algorithms:

- \( \text{KEYGENERATION}(\ell) \) generates a pair of keys (secret and public) \((SK, PK)\) for a given security parameter \( \ell \).
- \( \text{SIGN}(m, SK) \) receives the message \( m \) to be signed, the secret key \( SK \), and outputs the signature \( \sigma \).
- \( \text{VERIFY}(m, \sigma, PK) \) takes as input a message \( m \), signature \( \sigma \), and public key \( PK \). Using \( PK \), outputs 1 if the pair \((m, \sigma)\) is valid (as in Definition 2), and outputs 0 otherwise.

**Definition 2.** Let \( \Sigma \) be a CDSS as defined above, and let \((SK, PK)\) be a pair of secret and public keys. A pair of message and signature \((m, \sigma)\) is valid if \( \sigma \) is a valid output \(1\) of \( \Sigma.\text{SIGN}(m, SK) \).

Generally speaking, we say CDSS is unforgeable if a signature that verifies using \( PK \) can only be generated by the signer who has \( SK \). In more detail, we consider the model of adaptive chosen message attack, where the attacker \( A \) adaptively chooses a list of messages \( m_1, \ldots, m_q \), and requests the respective signatures \( \sigma_1, \ldots, \sigma_q \) from an oracle \( O \). Given this information, we say the attacker performs an existential forgery if he is able to produce a valid signature \( \sigma \) on a new message \( m \) chosen by him, \( m \neq m_i, 1 \leq i \leq q \) [16,22]. Because with unlimited time an adversary can perform all possible computations, we limit the computational power of the attacker by requiring an efficient algorithm to be one that runs in probabilistic polynomial time (PPT). Moreover, we say a function \( f \) is negligible if for every polynomial \( p \) there exists an \( N \) such that for all \( n > N \) we have that \( f(n) < 1/p(n) \) [14].

**Definition 3 (Unforgeability).** A digital signature scheme is existentially unforgeable under an adaptive chosen message attack if there is no PPT adversary \( A \) that can create an existential forgery with non-negligible probability.

3.2 Description of MTSS

Now we define the algorithms of the modification tolerant signature scheme (MTSS). We assume that the message \( m \) to be signed has size \(|m|\) in bits, and is

\footnote{We use “valid output” instead of \( \Sigma.\text{SIGN}(m, SK) = \sigma \) because the signing algorithm does not need to be deterministic.}
split into \( n \) blocks, not necessarily of the same size. A message split into blocks is represented as a sequence \( m = (m[1], \ldots, m[n]) \), where \( m[i] \) represents the \( i \)th block of message \( m \). For two messages \( m \) and \( m' \), each split into \( n \) blocks, we denote \( \text{diff}(m, m') = \{i \in \{1, \ldots, n\} : m[i] \neq m'[i]\} \). An \textit{authorized modification structure} is a collection \( S \subseteq P(\{1, \ldots, n\}) \), where \( P \) is the power set of \( \{1, \ldots, n\} \), that contains each set of blocks that the signer allows to be modified. The idea is that if \( \sigma \) is a valid signature for \( m \), then \( \sigma \) is a valid signature for \( m' \) if and only if \( \text{diff}(m, m') \in S \). Of course, to prevent inconsistencies we must have \( \emptyset \in S \). Indeed, an MTSS is equivalent to a CDSS if \( S = \{\emptyset\} \). A more general example is \( S = \{\emptyset, \{2\}, \{3\}, \{5\}, \{2, 3\}, \{2, 5\}, \{3, 5\}\} \) for \( n = 5 \) blocks, which specifies blocks 1 and 4 cannot be changed and any change of at most two other blocks is allowed. The authorized modification structure is used to provide flexibility of modifications while providing control for the signer. In practice, we do not expect \( S \) to be stored explicitly, but instead to be implicitly enforced by the scheme.

**Definition 4.** A modification tolerant signature scheme (MTSS) for authorized modification structure \( S \subseteq P(\{1, \ldots, n\}) \) on messages with \( n \) blocks is a tuple \( \Sigma \) of three algorithms:

- \( \text{MTSS-KeyGeneration}(\ell) \) generates a pair of secret and public keys \((SK, PK)\) for a given security parameter \( \ell \).
- \( \text{MTSS-Sign}(m, SK) \) receives the message \( m \) to be signed, the secret key \( SK \), and outputs the signature \( \sigma \).
- \( \text{MTSS-Verify}(m, \sigma, PK) \) takes as input a message \( m \), signature \( \sigma \), and public key \( PK \). Outputs \((1, I)\) if \((m, \sigma)\) is valid for modification set \( I \) (as in Definition 3), and outputs \(0\) otherwise.

**Definition 5.** Let \( \Sigma \) be an MTSS for authorized modification structure \( S \subseteq P(\{1, \ldots, n\}) \), and let \((SK, PK)\) be a pair of secret and public keys. A pair \((m, \sigma)\) of message and signature is valid if there exists \( m' \) such that \( \sigma \) is a valid output \(^2\) of \( \Sigma.\text{MTSS-Sign}(m', SK) \) and \( \text{diff}(m, m') \in S \). In this case, we say \((m, \sigma)\) is valid for modification set \( I \) (where \( I = \text{diff}(m, m') \)).

The definition of unforgeability of an MTSS scheme is exactly like Definition 3 but the existential forgery now needs to produce a valid signature as given in Definition 5. This is in alignment with the notions introduced in [6].

**Definition 6.** An MTSS \( \Sigma \) for authorized modification structure \( S \subseteq P(\{1, \ldots, n\}) \) has correction capability if it is a tuple \( \Sigma \) of four algorithms, which, in addition to the algorithms in Definition 4, has the following algorithm:

- \( \text{MTSS-Verify\&Correct}(m, \sigma, PK) \): takes as input a message \( m \), signature \( \sigma \), and public key \( PK \). Outputs a pair \((\text{ver}, m')\) where:
  1. \( \text{ver} = \Sigma.\text{MTSS-Verify}(m, \sigma, PK) \).

\(^2\) We use “valid output” instead of \( \Sigma.\text{MTSS-Sign}(m', SK) = \sigma \) because the signing algorithm does not need to be deterministic.
2. \( m' \) is a message with \( m' \neq \lambda \) (the corrected message) if \( \text{ver} = (1, I), I = \text{diff}(m, m') \), and \( (m', \sigma) \) is a valid pair for modification set \( I' = \emptyset \); in all other cases \( m' = \lambda \), which indicates failure to correct.

Location of modifications with MTSS would be trivial for any \( S \subseteq P(\{1, \ldots, n\}) \) if the signer simply produced \( \sigma \) as a tuple of \( n \) signatures, one for each block. However, this would be extremely inefficient for a large number of blocks \( n \). We must reconcile the objectives of having a large \( S \) and having a compact signature. Of course, the signature size depends on the security parameter, but once this is fixed, we would like the signature to grow moderately as a function of \( n \). This motivates the following definition of compression ratio.

**Definition 7.** An MTSS \( \Sigma^n \) for messages with \( n \) blocks and signature \( \sigma \) with \( |\sigma| \leq s(n) \) has compression ratio \( \rho(n) \) if \( \frac{n}{\rho(n)} \) is \( O(\rho(n)) \).

The compression ratio measures how efficient our signature is with respect to the trivial scheme of keeping one signature per block, with \( \rho(n) = O(1) \), supporting \( S = P(\{1, \ldots, n\}) \). Classical signatures have \( \rho(n) = n \) (best possible), but \( S = \{\emptyset\} \). In the next section we present a tradeoff, where \( \rho(n) = \frac{n}{\log n} \) and \( S \) is the set of all sets of up to \( d \) modifications, for fixed \( d \). Indeed, when using cover-free families it is possible to have a compression ratio of \( O\left(\frac{n^{d^2 \log \log n}}{d^2 \log n}\right) \) using known CFF constructions [19], while a lower bound on \( s(n) \) [10] tells us we cannot have \( \rho(n) \) larger than \( \Theta\left(\frac{n}{(d^2 / \log d \log n)}\right) \).

**4 d-Modification Tolerant MTSS Based on Combinatorial Group Testing**

Here we propose an MTSS that allows the modification of any set of up to \( d \) blocks in the message, for a constant \( d \) which we call a tolerance level. In other words, the authorized modification structure is \( S = \{T \subseteq \{1, \ldots, n\} : |T| \leq d\} \). To obtain a compact signature size, we rely on combinatorial group testing, which we summarize in Section 4.1 before we describe the scheme in Section 4.2. Similar modification location methods based on combinatorial group testing have been proposed in [11,12], and the instantiation we propose for the first three algorithms of MTSS (Section 4.2) are based on [12]. Our new contributions in this section include proof of security for the MTSS scheme based on cover-free families and the addition of error correction capability by proposing algorithm MTSS-VERIFY&CORRECT that corrects modifications in this context.

**4.1 Cover-Free Families and Group Testing**

Combinatorial group testing deals with discovering \( d \) defective items in a set of \( n \) items, via testing various groups of items for the presence of defects in each group. In nonadaptive combinatorial group testing, the groups are determined before the testing process starts. For the problems considered in this paper, a
modified block is a defective item, and the groups are sets of blocks combined and hashed together. In our case, we must use nonadaptive combinatorial group testing, as the tests are generated at time of signing, while verification is done later. Cover-free families allow for a small number of groups that help to identify the modified blocks.

Recall that a permutation matrix of dimension $l$ is an $l \times l$ matrix that is obtained from permuting rows of the identity matrix.

**Definition 8.** A $d$-cover-free family ($d$-CFF) is a $t \times n$ binary matrix $M$ such that any set of $d + 1$ columns contains a permutation submatrix of dimension $d + 1$.

Each column of $M$ corresponds to a block of the message, and each row corresponds to a group of blocks that will be tested together. A test fails if a group contains a modified block and passes if every block in a group is unchanged. The definition of $d$-CFF guarantees that for any column index $j$ and any other set of $d$ column indices $j_1, \ldots, j_d$, there will be a row $i$ s.t. $M_{i,j} = 1$, and $M_{i,j_1} = \cdots = M_{i,j_d} = 0$. In other words, for any unchanged block, there exists a test that contains that block but none of the up to $d$ modified blocks.

Figure 1 shows an example of how to use a 2-CFF(9, 12) to test a message with 12 blocks (represented by the columns) by testing 9 groups of blocks (the rows). Every unchanged block is in at least one group that passes the test (with result 0), and every group that failed the test (result 1) contains at least one modified block, which in this example are blocks 3 and 12. We note that column “result” is the bitwise-or of the columns corresponding to modified blocks 3 and 12.

| blocks | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|--------|---|---|---|---|---|---|---|---|---|----|----|----|
|        | ✓ | ✓ | X | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |    |    |    |
| $t_1$  | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 1  | 0  | 0  |
| $t_2$  | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0  | 1  | 0  |
| $t_3$  | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0  | 0  | 1  |
| $t_4$  | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 1  | 0  | 0  |
| $t_5$  | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0  | 1  | 1  |
| $t_6$  | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 1  | 0  | 0  |
| $t_7$  | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 0  | 0  | 1  |
| $t_8$  | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 1  | 0  | 0  |
| $t_9$  | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0  | 1  | 0  |

**Fig. 1.** Example of a 2-CFF(9, 12).

The next theorem is important for the efficiency of MTSS-VERIFY&CORRECT. Indeed, it ensures that for each modified block, we can find a test that contains it together with only other unchanged blocks. Therefore, an exhaustive trial-and-error can be used to guess this block until the hash of this group of blocks matches the original hash.
Theorem 1. Let $j_1, \ldots, j_d$ be the column indices that represent invalid elements. There is a row $i$ such that $M_{i,j_1} = 1, M_{i,j_2} = \ldots = M_{i,j_d} = 0$.

Proof. Since $M$ is a $d$-CFF, it is also a $(d - 1)$-CFF, therefore one of the rows in the permutation submatrix indexed by $j_1, \ldots, j_d$ is as stated.

4.2 Description of $d$-Modification Tolerant Signature Scheme

The main idea of a $d$-modification tolerant signature scheme is to sign a message split into blocks by concatenating the hashes of these blocks according to a $d$-CFF matrix. This allows us to locate up to $d$ modified blocks in the signed message, and correct these modifications. In this context, we represent a concatenation of two strings $a$ and $b$ as $a||b$, and $\lambda$ represents an empty string.

Definition 9. A $d$-modification tolerant signature scheme ($d$-MTSS) is an MTSS with authorized modification structure $S = \{T \subseteq \{1, \ldots, n\} : |T| \leq d\}$.

We now give an instantiation of $d$-MTSS using $d$-cover-free families based on [12].

Scheme 1: A $d$-Modification Tolerant Signature Scheme

The scheme requires an underlying CDSS $\Sigma$, a public hash function $h$, and a $d$-CFF$(t, n)$ matrix $M$. The algorithms are given next:

- MTSS-KeyGeneration($\ell$): generates a key pair $(SK, PK)$ using algorithm $\Sigma$.KeyGeneration($\ell$).
- MTSS-sign($m, SK$): Takes as input a secret key $SK$ and a message $m = (m[1], \ldots, m[n])$, and proceeds as follows.
  1. Calculate $h_j = h(m[j]), 1 \leq j \leq n$.
  2. For each $1 \leq i \leq t$, compute $c_i$ as the concatenation of all $h_j$ such that $M_{i,j} = 1, 1 \leq j \leq n$. Set $T[i] = h(c_i)$.
  3. Compute $h^* = h(m)$ and set $T = (T[1], T[2], \ldots, T[t], h^*)$.
  4. Calculate $\sigma' = \Sigma.Sign(T, SK)$. Output signature $\sigma = (\sigma', T)$.
- MTSS-verify($m, \sigma, PK$): takes as input a message $m = (m[1], \ldots, m[n])$, signature $\sigma = (\sigma', T)$ for $T = (T[1], T[2], \ldots, T[t], h^*)$, and public key $PK$, and proceeds as follows.
  1. Ensure that $\Sigma.Verify(T, \sigma', PK) = 1$, otherwise stop and output $(0, -)$.
  2. Check if $h^* = h(m)$. Stop and output $(1, \emptyset)$ if that is the case, continue otherwise.
  3. Use $M$ and $m$ and do the same process as in steps 1 and 2 of MTSS-sign to produce hashes $T'[1], \ldots, T'[t]$.
  4. Start with an empty set $V$, and for each $1 \leq i \leq t$ such that $T[i] = T'[i]$, compute the set of indices of unmodified blocks $V_i = \{j : M_{i,j} = 1\}$, and accumulate these values in the set of all indices of unmodified blocks $V = V \cup V_i$. Compute $I = \{1, \ldots, n\} \setminus V$. If $|I| \leq d$ output $(1, I)$, else output $(0, I)$. 
The correctness of the scheme is shown next.

**Theorem 2.** Consider a valid signature $\sigma$ generated by MTSS-Sign for a message $m$ and key pair $(SK, PK)$, and let $m'$ be a possibly modified message with $|\text{diff}(m, m')| \leq d$. Then, MTSS-Verify$(m', \sigma, PK) = (1, \text{diff}(m, m'))$.

**Proof.** Since $\sigma$ is valid, MTSS-Verify$(m', \sigma, PK)$ does not stop at step 1. If $m = m'$, then $h(m) = h(m')$ and the algorithm will stop in step 2, with $(1, \text{diff}(m, m') = \emptyset)$. It remains to check the case it stops in step 4. The $d$-CFF property guarantees that if a block has not been modified, it is contained in at least one valid concatenation with $T[i] = T'[i]$, since there is a row $i$ that avoids all modified blocks. Therefore, this block is contained in $V_i$. For each row $i$ that contains a modified block, we have $T[i] \neq T'[i]$, so modified blocks are not contained in any $V_i$. Therefore $I$ consists of precisely the modified blocks. Thus, $|I| \leq d$, and the algorithm outputs $(1, I = \text{diff}(m, m'))$. $\square$

The next theorem shows that when step 4 outputs $(0, I)$, Scheme 1 may give more information than required in Definition 4, as it may identify some unmodified blocks, even if not all.

**Theorem 3.** Consider a valid signature $\sigma$ generated by MTSS-Sign for a message $m$ and key pair $(SK, PK)$, and let $m'$ be a modified message with $|\text{diff}(m, m')| > d$. Then, MTSS-Verify$(m', \sigma, PK) = (0, I)$, and for any $i \in \{1, \ldots, n\} \setminus I$, block $m[i]$ is guaranteed to be unmodified.

**Proof.** This case will lead MTSS-Verify to step 4, and since $|\text{diff}(m, m')| > d$, the output will be $(0, I)$. Any block in $\{1, \ldots, n\} \setminus I$ is guaranteed to be part of matching row $i$, and must be unmodified, even though not every unmodified block will necessarily be placed in $\{1, \ldots, n\} \setminus I$. $\square$

Scheme 1 has been proposed in [12]. One of our main contributions here is to add correcting capability to $d$-MTSS. We require the size of each block to be upper bounded by a value $s$ that is small enough that guessing each of the (up to) $d$ modified blocks is “computationally feasible”. Basically, by brute force we compute up to $O(d2^s + n)$ hashes to accomplish the modification correction (see the algorithm under Scheme 2). We use the indices of the modified blocks in $I$ and do an exhaustive search to recover their original values.

**Scheme 2: A $d$-MTSS with Correction Capability**

The scheme requires an underlying CDSS $\Sigma$, a public hash function $h$, and a $d$-CFF$(t, n)$ matrix $M$. It further requires that the message is divided into $n$ blocks of size at most $s$. The scheme has the three algorithms from Scheme 1, and additionally the algorithm below:

- **MTSS-VERIFY&CORRECT$(m, \sigma, PK)$:** receives as input a message $m = (m[1], \ldots, m[n])$, a signature $\sigma = (\sigma', T)$ where $T = (T[1], T[2], \ldots, T[t], h^*)$, and a public key $PK$, and proceeds as follows.
1. Compute \( \text{result} = \text{MTSS-verify}(m, \sigma, PK) \). If \( \text{result} = (0, X) \), then stop and output \((0, X)\), otherwise \( \text{result} = (1, I) \). If \(|I| = 0 \) go to step 6, otherwise run steps 2 – 5 for each \( k \in I \).

2. Identify a row \( i \) in \( M \) such that \( M_{i,k} = 1 \) and \( M_{i,j} = 0 \), for all \( j \in I \setminus \{k\} \).

3. Compute \( h_j = h(m[j]) \) for all \( j \) such that \( M_{i,j} = 1, j \neq k \), \( i \) from step 2. Set \( \text{corrected}[k] = \text{false} \).

4. For every possible binary string \( b \) of size \( \leq s \), proceed as follows:
   - Compute \( h_k = h(b) \).
   - For \( i \) obtained in step 2 and \( 1 \leq j \leq n \), compute \( c_i \) as the concatenation of every \( h_j \) such that \( M_{i,j} = 1 \) and set \( T'[i] = h(c_i) \).
   - If \( T'[i] = T[i] \) and \( \text{corrected}[k] = \text{false} \), set \( \text{corrected}[k] = \text{true} \) and correct the block \( m[k] = b \).
   - Else, if \( T'[i] = T[i] \) and \( \text{corrected}[k] = \text{true} \), stop and output \((1, I, \lambda)\).

5. Return to step 2 with the next \( k \in I \).

6. Output \((1, I, m)\).

We note that the flag \( \text{corrected}[k] \) is used to identify a possible collision of two different bit strings \( b \) giving the same hash value \( h(m[k]) \). Since the correct block cannot be determined, we exit with \( \lambda \) indicating failure. The next proposition has details on the correctness of this algorithm.

**Proposition 1.** Let \((m, \sigma)\) be a valid pair of message and signature produced by \( \text{MTSS-Sign}(m, SK) \), using a hash function \( h \) and with \( m = (m[1], \ldots, m[n]) \). Let \( m' \) be a message and let \( I = \text{diff}(m, m') \) with \(|I| \leq d \). If for every \( k \in I \), \( h(m[k]) \) has no other preimage of size up to \( s \), then \( \text{MTSS-Verify&Correct}(m', \sigma, PK) = (1, I, m) \).

**Proof.** As seen in Theorem 2, the set \( I \) contains precisely the indices of the modified blocks, and Theorem 1 guarantees that such a row \( i \) in step 2 of the algorithm exists. Finally, if for every \( k \in I \) the hash \( h(m[k]) \) has no second preimage, then step 4 of the algorithm computes a unique replacement for each modified block, and the algorithm outputs the corrected message in step 6. \( \square \)

Now we prove that when selecting a good hash function, we can always guarantee the correction of any up to \( d \) modified blocks.

**Theorem 4.** Consider Scheme 2 restricted to messages with blocks of size at most \( s \), and such that \( h \) is a hash function where no two inputs of size up to \( s \) have the same hash value. Then, \( \text{MTSS-Verify&Correct}(m', \sigma, PK) \) can always correct a message with up to \( d \) modified blocks.

**Proof.** Easily obtained by Proposition 1 since no matter what is the value of the block, no other block of size up to \( s \) can have the same image under \( h \). \( \square \)

Next, we show that the assumption of existence of such hash function is realistic, and in fact very easy to find.
Proposition 2. Consider a family of hash functions \( h : X \rightarrow Y \) where \( |Y| = 2^l \) and a subset of the inputs \( S \subseteq X \) where \( |S| \leq 2^{s+1} \). The probability that there is a collision in \( S \), i.e. the probability that there exists \( x, z \) in \( S \) with \( h(x) = h(z) \), is approximately \( \epsilon = 1 - e^{-2^{s+1}} \).

Proof. This comes from the fact that the probability of finding at least one collision after \( Q \) hash calculations is approximately \( 1 - e^{-Q/2^{2l+1}} \), and in our application \( Q = |S| = 2^{s+1} \).

In practice, we will set \( 2s \ll l \). This ensures via Proposition 2 that \( h \) is very likely to have the desired property of being injective on \( S \), the set of binary strings with size at most \( s \). In the unlikely event that \( h \) fails this property, we try another hash function until we succeed. The expected number of attempts will be very close to 1. For example, if we consider SHA-256 as the hash function, \( |Y| = 2^{256} \), and for \( s = 20 \), the probability of a collision within the set of size at most 20 is \( \epsilon = 1 - e^{-2^{40-256+1}} \approx 3.70 \times 10^{-68} \). Indeed, we experimentally verified that SHA-256 has no collisions for \( s = 20 \).

Theorem 5. Consider Scheme 2 with tolerance level \( d \), and let \( m \) be a message split into \( n \) blocks, each block of size at most \( s \). Let \( \sigma = \text{MTSS-Sign}(m, SK) \) and \( m' \) be a message with \( |I| = \text{diff}(m, m') \) with \( |I| \leq d \). Then, \( \text{MTSS-Verify\&Correct}(m', \sigma, PK) \) returns \((1, I, m)\) and uses \( O(n + d^2 \log n + d2^s) \) hash computations.

Proof. By choosing a suitable hash function, Theorem 2 guarantees that \( \text{MTSS-Verify\&Correct} \) returns \((1, I, m)\). The algorithm starts with the location of the modified blocks. This step uses a \( d \)-CFF for which we can use a construction with \( t = O(d^2 \log n) \) rows \[19\], and therefore a total of \( n + t \) hash calculations are required to locate these modifications. After locating up to \( d \) modified blocks, we need to perform the following computations for each one of them. We compute every possible block of size up to \( s \) and their corresponding hash values (total of \( 2^{s+1} - 1 \)), and according to the row of the CFF matrix, we compute a few extra hashes (in total not more than \( n \), if storing \( h_i \) instead of recomputing among different runs of of line 3). This gives a total of \( O(d2^s + n) \) hash computations for the correction step.

4.3 Comparing Scheme 2 with Scheme E

Let an \((l, n, D)_q\)-ECC \( \mathcal{C} \) be an error correcting code with minimum distance \( D \), codewords of length \( l \) that encode messages of size \( n \) over an alphabet of size \( q \). The alphabet must be so that it can distinguish each possible block considered in Scheme 2, so \( q \geq 2^s \). For simplification, assume \( c_m = (m, b_m) \) where \( b_m \) is a tuple of \( l - n \) letters ("check bits"). This code can correct \( d = \lfloor \frac{D-1}{2} \rfloor \) errors (modifications in the message). Next, we describe signature and verification using Scheme E. Using the same inputs as \( \text{MTSS-Sign} \), algorithm \( \text{ECC-Sign}(m, SK) \) do the following steps: 1) Compute \( \sigma' = \Sigma . \text{Sign}(m, SK) \);
2) Compute $c_m = (m, b_m)$ according to ECC $C$ and return $\sigma = (b_m, \sigma')$. Then, algorithm ECC-VERIFY&CORRECT($m, \sigma = (b, \sigma'), PK$) do the following steps: 1) Decode $c = (m, b)$ to $m'$ using $C$; 2) If $\Sigma.VERIFY(m', \sigma, PK)) = 1$ then return $(1, \text{diff}(m, m'), m')$, else return $0$.

Since $C$ can correct up to $d$ errors, if the signature $\sigma$ is valid ($\sigma'$ is authentic for $m'$ and $|\text{diff}(m, m')| \leq d$), then ECC-VERIFY&CORRECT will behave in the same way as MTSS-VERIFY&CORRECT and will return $(1, \text{diff}(m, m'), m')$. However, when ECC-VERIFY&CORRECT returns $0$, it does not distinguish between the two failing cases obtained by MTSS-VERIFY&CORRECT, namely:  

**Case 1)** output $(0, -)$, which means the CDSS signature $\sigma'$ is not authentic;  

**Case 2)** output $(0, I)$, which means $\sigma'$ is authentic and message $m$ differs from the signed $m'$ in more than $d$ blocks, and also if $|I| < n$, then $|\{1, \ldots, n\} \setminus I| > 0$ and we are sure of at least one unmodified block. Therefore, while the Scheme E is an MTSS scheme according to Definition 4, it provides less information than Scheme 2.

A comparison of Scheme 2 with Scheme E shows they have similar compression ratios. Indeed, we first note that given an $(l, n, D)q$-ECC, it is possible to construct a $d$-CFF$(l \cdot q, q^n)$, with $d = \lfloor (l-1)/(l-D) \rfloor$ [16]. Then, we consider some families of error correcting codes, and for each family we restrict Scheme 2 to only use CFFs constructed using codes from this family. In the next table, we summarize the compression ratios obtained for 3 families of codes, but we omit the details of our calculations due to lack of space.

| Code                  | Scheme 2       | Scheme E       |
|-----------------------|----------------|----------------|
| Hamming, $n = 2^{2^t-1}$ | $\approx 2^{2^t-2r-2}$ | $\approx 2^{2^t-2r-1}$ |
| Reed-Solomon, $n = q^{\frac{(q-1)t}{2}}-1$ | $q^{\frac{(q-1)t}{2}}-1$ | $q^{\frac{(q-1)t}{2}}+1$ |
| PR-code [19]          | $\Theta(\frac{n}{d^2 \log n})$ | $O(\frac{n}{d^2 \log n})$ |

In conclusion, both schemes serve the same purpose with similar compression ratios, but Scheme 2 has several advantages. First, Scheme 2 gives more information in the failing case where ECC returns $0$, as discussed above. Second, Scheme 2 provides a non-correcting version of the verification algorithm (MTSS-VERIFY) which in the case of unmodified messages is basically as time efficient as $\Sigma.VERIFY$ (see step 1 of MTSS-VERIFY); in this case, Scheme E still needs to run a decoding algorithm, with complexity influenced by $q \geq 2^s$, where $s$ is the largest size of a block. Finally, Scheme 2 is part of a family of similar schemes presented here (Schemes 1-3), which can be more easily compared.

### 5 Security

In this section, we present the security proof of $d$-MTSS for Schemes 1 and 2. In order to do this, we need to check that the security of the hash function $h$ and the unforgeability of the underlying CDSS can together ensure unforgeability of $d$-MTSS. Note that although Scheme 1 appeared in [12], no security proof has been given in that paper. For the next proof we assume a collision-resistant hash function, i.e. a hash function in which a collision cannot be efficiently found [22].
When we consider $d$-MTSS using $d$-CFFs, a valid $(m, \sigma)$ as in Definition 5 implies that there exists $m'$ such that $\sigma$ is an output of MTSS-Sign$(m', SK)$ and $|\text{diff}(m, m')| \leq d$. In the next theorem we suppose there is a valid $(m, \sigma)$ as a forgery to our scheme. We consider two types of forgery: a strong forgery consists of $(m, \sigma)$ such that MTSS-Verify$(m, \sigma, PK) = (1, I), |I| \leq d$; a weak forgery consists of $(m, \sigma)$ such that MTSS-Verify$(m, \sigma, PK) = (1, \emptyset)$.

**Theorem 6.** Let $X$ be a $d$-MTSS as described in Scheme 1 based on an existentially unforgeable CDSS $\Sigma$ and on a collision resistant hash function $h$. Then, $X$ is existentially unforgeable.

**Proof.** (By contradiction) Suppose $X$ is not existentially unforgeable. Then, there is an adversary $A$ that, after $q$ adaptive queries to a signing oracle $O$, obtains pairs $(m_1, \sigma_1), \ldots, (m_q, \sigma_q)$, with $\sigma_i = (\sigma'_i, T_i), 1 \leq i \leq q$, and with non-negligible probability, outputs a valid pair $(m, \sigma)$, with $\sigma = (\sigma', T)$, $T = (T[1], T[2], \ldots, T[|I|], h^*)$, and $|\text{diff}(m, m)| > d$, for all $1 \leq i \leq q$.

We show that if such $A$ exists, then we can build a probabilistic polynomial time algorithm $A'$ which has the following input and output:

**Input:** an existentially unforgeable CDSS $\Sigma$ and a collision-resistant hash function $h$, both with security parameter $\ell$.

**Output:** either a existential forgery $(T, \sigma')$ of $\Sigma$ or a collision pair for $h$.

Next we describe the steps of such $A'$.

1. Simulate the probabilistic polynomial time adversary $A$ forging an MTSS based on $\Sigma$ and $h$ using queries mentioned above. With non-negligible probability, this will produce a forgery $(m, \sigma = (\sigma', T))$ in $X$, as described above. Otherwise, return “FAIL”.
2. If $T \neq T_i$, for all $1 \leq i \leq q$, $A'$ presents $(T, \sigma')$ as a forgery in $\Sigma$, for $\sigma'$ the corresponding signature of $T$.
3. If $T = T_i$, for some $1 \leq i \leq q$, first calculate $I$ by computing MTSS-Verify$(m, \sigma, PK)$. Then, we have:
   - In the case of weak forgery, we must have $I = \emptyset$, and so the final element of $T$ is $h(m)$ and the final element of $T_i$ is $h(m_i)$, and since $m \neq m_i$, $A'$ presents $(m, m_i)$ as a collision pair for $h$.
   - Otherwise, we must have a strong forgery with $1 \leq |I| \leq d$. So, there exists $m'$ such that MTSS-Verify$(m', \sigma, PK) = (1, \emptyset)$ and $|\text{diff}(m, m')| \leq d$. Since $|\text{diff}(m, m_i)| > d$, there must be a block $m[k], k \in \{1, \ldots, n\} \setminus I$, that is considered valid in $m$ but $m[k] \neq m_i[k]$. Let $p$ be any row of the CFF matrix $\mathcal{A}$ with $\mathcal{M}_{p,k} = 1$. Because $T = T_i$, this implies $T[p] = h(c) = h(c') = T_i[p]$, for $c = h(m[j_1])||h(m[j_2])|| \ldots ||h(m[j_s])$ and $c' = h(m_i[j_1])||h(m_i[j_2])|| \ldots ||h(m_i[j_s])$, where $k \in \{j_1, j_2, \ldots, j_s\}$. If $h(m[k]) = h(m_i[k])$, then $A'$ presents $(m[k], m_i[k])$ as a collision pair for $h$. Otherwise, $A'$ presents $(c, c')$ as a collision pair for $h$.

The probability $p(\ell)$ that $A'$ succeeds is the same as the probability that $A$ succeeds, where $\ell$ is the security parameter. Whenever $A'$ succeeds, we have
either an existential forgery of $\Sigma$ or a collision for $h$, corresponding to steps 2 and 3, respectively. For each $\ell$, one of these steps is at least as likely to occur as the other, so it has probability at least 1/2. So, for any security parameter $\ell$, use one of two algorithms (an existential forger for $\Sigma$, or a collision finder for $h$) that runs in probabilistic polynomial time and succeeds with probability at least $p(\ell)/2$. In other words, we can technically design two adversaries $A'_1$ and $A'_2$ based on $A'$. $A'_1$ forges $\Sigma$ by proceeding as $A'$ but returning “FAIL” if it falls in the case of step 3. Similarly, $A'_2$ finds a collision for $h$ or returns “FAIL” if it falls in the case of step 2. Then, either $A'_1$ or $A'_2$ will succeed with probability at least $p(\ell)/2$ for infinitely many $\ell$. This contradicts either the unforgeability of $\Sigma$ or the collision resistance of $h$. \qed

6 Using MTSS for Redactable Signatures

Now we turn our attention to using MTSS in general and using similar algorithms to our proposed $d$-MTSS for redactable signature schemes. Redactable and sanitizable signature schemes allow for parts of a message to be removed or modified while still having a valid signature without the intervention of the signer. Sanitizable signatures usually requires the existence of a semi-trusted party called the sanitizer who is entrusted to do the modifications and recomputation of the signature, in some schemes done in a transparent way (see [18]). Our proposed scheme does not deal with sanitizable, but rather with redactable signatures.

Redactable signatures have been proposed in several variants also under the names of content extraction signatures [21] and homomorphic signatures [13]. Redactable signature schemes (RSS) “allow removing parts of a signed message by any party without invalidating the respective signature” [6] and without interaction with the signer. In [21], the authors mention content extraction for privacy protection or bandwidth savings. Suppose Bob is the owner of a document signed by Alice and does not want to send the whole document to a third verifying party Cathy but only some extracted parts of the document; however, Cathy still needs to verify that Alice is the signer of the original document. Alice is agreeable with future redactions when she signed the document. An example given in [21] is that Bob has his university transcripts signed by the issuing university (Alice) and wants to submit the transcripts to a prospect employer (Cathy) without revealing some of his private information such as his date of birth. Cathy must be able to verify that the signature came from the university in spite of the redaction. In addition to the privacy application, content extraction can be used in a similar way when only part of a large document needs to be passed by Bob to Cathy for the purpose of reducing the communication bandwidth [21].

The notion of redactable signatures we consider next is in line with our general definition of MTSS, but differs in that we add another algorithm called MTSS-REDACT and we require total privacy of the redacted parts. We give next a $d$-MTSS version that is redactable based on ideas similar to Scheme 1 but
modifying it to guarantee total privacy of redacted blocks. As mentioned in [18],
the scheme proposed in [12] which was presented as Scheme 1 does not meet
standard privacy requirements and in particular leaks the original message’s hash
value. The scheme we propose below addresses this issue by individually signing
each of the hashes of the groups of blocks, and at the time of redaction of blocks,
also redacting from the signature tuple any hashes involving concatenations of
the modified blocks. To avoid more complex forms of forgery, that could take
advantage of combining individual signatures of concatenations coming from
different signed messages, we add the same random string to the various hashes
in addition, to avoid reordering of blocks within the same message via reordering
of concatenations to link the individual parts that are signed at the same time;
in addition, to avoid reordering of blocks within the same message via reordering
the groups, we add a group counter to these hashes before signing.

In the description below, a redaction is represented by the symbol ■.

Scheme 3: A Redactable d-MTSS with Privacy Protection

The scheme requires an underlying CDSS $\Sigma$, a public hash function $h$, a
d-CFF($t,n$) matrix $\mathcal{M}$ and a random number generator $\text{RAND}$. The algorithms are given next:

\begin{itemize}
  \item MTSS-KeyGeneration($\ell$): generates a key pair ($SK, PK$) using algo-
  rithm $\Sigma$.KeyGeneration($\ell$).
  \item MTSS-sign($m, SK$): Takes as input $SK$ and a message $m = (m[1], \ldots, m[n])$,
  and proceeds as follows.
  \begin{enumerate}
    \item Calculate $h_j = h(m[j]), 1 \leq j \leq n$. Compute a random string $r = \text{RAND}()$.
    \item For each $1 \leq i \leq t$, compute $c_i$ as the concatenation of all $h_j$ such that
      $\mathcal{M}_{i,j} = 1, 1 \leq j \leq n$, and set $T[i] = h(c_i)||id(i, t+1)$, where $id(i, t+1)$
      encodes the numbers $i$ and $t+1$.
    \item Compute $h^* = h(m)$, set $T[t+1] = h^*||id(t+1, t+1)$ and set $T = (T[1], \ldots, T[t+1])$.
    \item Calculate $\sigma'[i] = \Sigma.\text{Sign}(T[i], SK)$, for each $1 \leq i \leq t+1$ and set
      $\sigma' = (\sigma'[1], \sigma'[2], \ldots, \sigma'[t+1])$. Output signature $\sigma = (\sigma', r)$.
  \end{enumerate}

  \item MTSS-verify($m, \sigma, PK$): takes as input a message $m = (m[1], \ldots, m[n])$,
  a signature $\sigma = (\sigma', r)$, and a public key $PK$, and proceed as follows.
  \begin{enumerate}
    \item Check if $\Sigma.\text{Verify}(h(m)||r||id(t+1, t+1), \sigma'[t+1], PK) = 1$. Stop and output (1, 0) if that is the case; continue otherwise.
    \item Use $\mathcal{M}, m, r$ and do the same process as in steps 1-3 of MTSS-sign to
      produce tuple $T' = (T'[1], \ldots, T'[t])$.
    \item For each $1 \leq i \leq t$ such that $\Sigma.\text{Verify}(T'[i], \sigma'[i], PK) = 1$, compute
      the set of indices of intact blocks $V_i = \{j: \mathcal{M}_{i,j} = 1\}$ and do $V = V \cup V_i$.
      Compute $I = \{1, \ldots, n\} \setminus V$. Output (1, $|I|$) if $|I| \leq d$; output 0 otherwise.
  \end{enumerate}

  \item MTSS-redact($m, \sigma, R$): takes as input a message $m = (m[1], \ldots, m[n])$,
  a signature $\sigma = (\sigma', r)$, and a set $R \subseteq \{1, \ldots, n\}$ of blocks to be redacted, with
  $|R| \leq d$, and proceeds as follows.
  \begin{enumerate}
    \item If $R = \emptyset$, then stop and output $(m, \sigma)$.
2. Create copies of the message and signature: $\overline{m} = m$, $\overline{\sigma} = \sigma$, so that $\overline{\sigma} = (\overline{\sigma'}, r)$.
3. Set $\overline{\sigma'}[t + 1] = \Box$.
4. For each $j \in R$: set $\overline{m}[j] = \Box$ and for each index $i$ such that $\mathcal{M}_{i,j} = 1$ set $\overline{\sigma'}[i] = \Box$.
5. Output $(\overline{m}, \overline{\sigma} = (\overline{\sigma'}, r))$.

The correctness of the first three algorithms follows similar reasoning using the CFF properties as argued in Theorem 2, taking into account the different approach used here where $t$ CDSS signatures and $t$ CDSS verifications are required. The redaction offers total privacy in the sense of information theory, as no information related to the redacted blocks is kept in the signature, which are erased in steps 3 and 4 of MTSS-REACT. The redacted blocks and redacted parts of the signature will only affect the verification of the parts of the signatures involving redacted blocks; all other block indices will be indicated as unmodified in the output of line 3 of MTSS-VERIFY, as long as no more than $d$ blocks have been redacted.

In the next theorem, we establish the unforgeability of this scheme.

**Theorem 7.** Let $X$ be a redactable $d$-MTSS given in Scheme 3 based on an existentially unforgeable CDSS $\Sigma$, on a collision-resistant hash function $h$, and on a random Oracle. Then, $X$ is existentially unforgeable.

**Sketch of the proof.** Similar reasoning as in Theorem 6 can be used to argue that the unforgeability of $\Sigma$ and the collision resistance of the hash function are sufficient to protect against forgery at the level of individual parts of the tuple signature. However, we need to rule out more complex forgery attempts that could try to combine individual signatures in the signature tuple in different ways as well as different blocks from different messages. The use of a common random string in the creation of each $T[i]$ in lines 2 and 3 of algorithm MLSS-SIGN prevents an adversary from trying to combine message blocks and signature parts of $(m_1, \sigma_1), \ldots, (m_q, \sigma_q)$ coming from different calls to the random Oracle to create a new valid signature since, by definition, two different calls to the oracle will, with high probability, not produce the same random string. Furthermore, the concatenation of the encoding of the test index and the number of tuple elements, $id(i, t + 1)$, within each tuple position $T[i]$ in line 2 and 3, makes it highly unlikely that blocks are added, removed or reordered, since this would amount to finding a collision in the hash function.

7 Implementation Issues

When implementing Schemes 1-3, there are a few details that need to be considered regarding efficiency, security, and even flexibility on the inputs or outputs of the algorithms. For example, we consider a message divided into blocks that we represented as a sequence, but blocks may be represented using different data structures depending on the application and the type of data. Pohls [18] discusses sequence, set and tree data structures for blocks, which could be employed.
When signing a message using MTSS-Sign as described in Schemes 1-3, we could consider not hashing every block before concatenation, especially if the sizes of the blocks are small enough that a hash function will end up increasing the size of the data being concatenated. This approach needs to be carefully considered, since a simple concatenation does not insert a delimitation on where the individual blocks start or end, and therefore blocks with a shared suffix or prefix may lead to wrong identifications of valid/invalid. To be safe, we can hash each block before concatenating them as presented in our schemes, or ensure the concatenations use delimiters to separate blocks, or require blocks of fixed size.

The correction algorithm MTSS-Verify&Correct computes the set of indices of modified blocks $I$ and tries to correct these blocks to their original value. If there is at least one position where two different blocks match the hash of the original one (a second preimage on the hash function), the algorithm aborts the correction process and outputs an empty string $\lambda$ to represent correction error. As seen in Proposition 2 and the discussion that follows, it is very unlikely for such event to occur, and we could always choose another hash function with no collisions for certain block sizes. However, if some specific application is not flexible in the choice of hash functions and such an event happens, an interesting approach can be to correct as many modified blocks as possible, and return some information regarding the ones with collisions (such as a fail message or even a list of possible values for those specific blocks). This approach allows for partial correction of modified data, that may be interesting for some applications. Moreover, if we already know that the chosen hash function has no collisions for blocks of size up to $s$, we do not need to run step 4 for every possible block of size $\leq s$, and we could stop this loop as soon as we find the first match, improving the efficiency of the method.

Regarding the multiple hash computations that happen in the correction algorithm, one could consider some improvements. Note that we already compute the hashes of all unmodified blocks when we do the call of MTSS-Verify to obtain the set $I$, so these hashes of unmodified blocks could be reused in step 2. Moreover, we repeat for every modified block in $I$ the same process of computing all blocks of size $\leq s$ and their corresponding hash values. For the cases where we have a big set of modified blocks (big $d$), one may consider to pre-compute all these values, in case the application can handle the storage that this will require.

8 Parameter Relations

For MTSS with modification location only (Scheme 1), a message $m$ is split into $n$ blocks of no specific size, and a location capability parameter $d$ needs to be decided. These parameters are used to construct a $d$-CFF$(t, n)$ with number of rows $t = \Theta(d^2 \log n)$ if using [19], $t = d\sqrt{n}$ if using [17], and $t = q^2$ when using [9], for any prime power $q$ and positive integer $k$ that satisfy $q \geq dk + 1$. The signature size is impacted by $d$ and $n$, since there are $t+1$ hashes as part of the signature. Therefore, while smaller sizes of blocks and larger $d$ give a more precise location of modifications, they also increase the size of the signature.
Now consider the exact same message $m$, but for the case of an MTSS with correction capabilities (Scheme 2). This scheme requires that the blocks have a small size of up to $s$ bits, which implies that the very same message $m$ now has many more blocks $n' \gg n$. A larger number of blocks directly increases the size of the signature. But now, the $d$ that was enough to locate the modifications before may not be enough anymore, since modifications that were in one big block before may now be spread over several small blocks. When locating the modifications, the algorithm aborts the process if the number of modified blocks is larger than the expected location capability $d$, which may cause the scheme to fail to correct more often if this value is not increased as $n'$ increases.

The size of the signature $\sigma$ in Schemes 1 and 2 is $|\sigma| = |h(.)| \times (t + 1) + |\sigma'|$, and in Scheme 3 is $|\sigma| = |\sigma'| \times (t + 1) + |r|$, for $\sigma'$ a classical digital signature, $r$ a random bit string, $h(.)$ a traditional hash function, and $t$ the number of rows of a $d$-CFF($t, n$). The input consists of a message of size $|m| = N$ in bits, divided into $n$ blocks, each of size at most $s$, so $N \approx n \times s$. In Scheme 2, given $N$, we need to wisely choose $s$ so it is small enough to allow corrections of blocks, while guaranteeing $n$ is small enough to have a reasonable $|\sigma|$. We cannot expect our signature to be as small as the ones from classical digital signature schemes, since we require extra information in order to be able to locate and correct portions of the data. In summary what we want is $n \times s \gg |\sigma|$.

The next examples show that even for small $s = 8$, we still have a reasonably small signature.

**Example 1:** $N = 1 \text{ GB} = 2^{30}$ bytes $= 2^{30}2^3 = 2^{33}$ bits, $h = \text{SHA-256}$, $s = \log |h| = \log 2^8 = 8$ bits, $d = 4$, and we use RSA with a 2048-bit modulus. Then we have $n = N/s = 2^{33}/2^3 = 2^{30}$ blocks, with 8 bits each. Consider the $d$-CFF($q^2, q^{k+1}$) from [9], with $k = 6, q = 25, t = 25^2, n = 25^7$. Now, since $|\sigma| = |h(.)| \times (t + 1) + |\sigma'|$ in Schemes 1 and 2, we have $|\sigma| = 2^8 \times (25^2 + 1) + 2048 = 162304$ bits, which is 20288 bytes, or $\sim 20$ KB.

**Example 2:** For the same message and parameters as in Example 1, now we use a random bit string $r$ of size 128 bits and create a signature as in Scheme 3. Since $|\sigma| = |\sigma'| \times (t + 1) + |r|$, we have $|\sigma| = 2048 \times (25^2 + 1) + 128 = 1282176$ bits, which is 160272 bytes, or $\sim 160$ KB.

For Scheme 3, small blocks like in Example 2 are not required, so we can get a much smaller signature. The choice of $s$ and $n$ in this case depends on the application, and on whether we wish to correct non-redactable blocks.

**9 Conclusion**

We introduce modification tolerant signature schemes (MTSS) as a general framework to allow localization and correction of modifications in digitally signed data. We propose a scheme based on $d$-CFFs that allows correction of modifications, and a variation without correction that gives redactable signatures. The presented schemes are provably secure and present a digital signature of size close to the best known lower bound for $d$-CFFs.
Interesting future research includes an implementation of the schemes proposed here, with practical analysis of parameter selection for specific applications. In addition, new solutions for MTSS beyond $d$-CFFs can also be of interest. The $d$-CFF treats every block the same and allows for any combination of up to $d$ blocks to be modified. Specific applications can have different requirements about what combinations of blocks can be modified by specifying a different authorized modification structure. Moreover, other hypotheses about more likely distribution of modifications throughout the document can be used for efficiency; for example, modified blocks may be concentrated close to each other. One idea in this direction is to use blocks with sub-blocks for increasing granularity of modification location and to aid correction. This would involve matrices with a smaller $d$ for the bigger blocks and a larger $d$ for sub-blocks of a block, picking these parameters with the aim of decreasing $t$ and consequently signature size. It is out of the scope of this paper to go into detailed studies of other types of modification location matrices, which we leave for future work.

References

1. G. Ateniese, D. H. Chou, B. de Medeiros, and G. Tsudik. Sanitizable Signatures. In European Symposium on Research in Computer Security (ESORICS 2005), Lecture Notes in Computer Science, 3679, pages 159–177.
2. P. S. L. M. Barreto, H. Y. Kim and V. Rijmen. Toward secure public-key blockwise fragile authentication watermarking. In IEE Proceedings - Vision, Image and Signal Processing, 149(2):57–62, 2002.
3. R. G. Biyashev and S. E. Nyssanbayeva. Algorithm for creating a digital signature with error detection and correction. Cybernetics and Systems Analysis, 48(4):489–497, 2012.
4. A. De Bonis and G. Di Crescenzo. Combinatorial group testing for corruption localizing hashing. In Computing and Combinatorics, pages 579–591, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.
5. A. De Bonis and G. Di Crescenzo. A group testing approach to improved corruption localizing hashing. Cryptology ePrint Archive, Report 2011/562, 2011. https://eprint.iacr.org/2011/562
6. D. Derler, H. C. Pöhl, K. Samelin, and D. Slamanig. A general framework for recraftable signatures and new constructions. In International Conference on Information Security and Cryptology (ICISC 2015), Lecture Notes in Computer Science, 9558, pages 3–19. Springer, 2015.
7. G. Di Crescenzo, R. Ge, and G. R. Arce. Design and analysis of dbmac, an error localizing message authentication code. In IEEE Global Telecommunications Conference, 2004, GLOBECOM'04., volume 4, pages 2224–2228, 2004.
8. P. D’Arco and D. Stinson. Fault tolerant and distributed broadcast encryption. In Proceedings of the 2003 RSA Conference on The Cryptographers’ Track, pages 263–280. Springer-Verlag, 2003.
9. P. Erdős, P. Frankl and Z. Füredi. Families of finite sets in which no set is covered by the union of r others. Israel J. Math., 51 (1985), 79–89.
10. Z. Füredi, On r-Cover-free Families, Journal of Combinatorial Theory, 73 (1996), 172–173.
11. M. T. Goodrich, M. J. Atallah, and R. Tamassia. Indexing information for data forensics. In *ACNS*, 2005.

12. T. B. Idalino, L. Moura, R. F. Custódio, and D. Panario. Locating modifications in signed data for partial data integrity. *Information Processing Letters*, 115(10):731–737, 2015.

13. R. Johnson, D. Molnar, D. Song, and D. Wagner. Homomorphic signature schemes. In *Topics in Cryptology - Cryptographers Track - RSA 2002.*, pages 244–262. Springer, 2002.

14. J. Katz, Y. Lindell. *Introduction to Modern Cryptography*. Chapman & Hall/CRC, 2007.

15. A. J. Menezes, S. A. Vanstone, and P. C. Van Oorschot. *Handbook of Applied Cryptography*. CRC Press, Inc., Boca Raton, FL, USA, 1st edition, 1996.

16. H. Niederreiter, H. Wang, and C. Xing. Function Fields Over Finite Fields and Their Applications to Cryptography. In: Garcia A., Stichtenoth H. (eds) *Topics in Geometry, Coding Theory and Cryptography*. Algebra and Applications, 2006, 59–104.

17. J. Pastuszak, J. Pieprzyk, and J. Seberry. Codes identifying bad signature in batches. In *INDOCRYPT*, volume 1977 of *Lecture Notes in Computer Science*, pages 143–154. Springer, 2000.

18. H. C. Pöhl. Increasing the Legal Probative Value of Cryptographically Private Malleable Signatures. Ph.D. Thesis, University of Passau, 2018.

19. E. Porat, A. Rothschild. *Explicit nonadaptive combinatorial group testing schemes*. IEEE Transactions on Information Theory 57 (2011) 7982–7989.

20. R. Safavi-Naini and H. Wang. New results on multi-receiver authentication codes. In Kaisa Nyberg, editor, *Advances in Cryptology — EUROCRYPT’98*, pages 527–541. Springer Berlin Heidelberg, 1998.

21. R. Steinfeld, L. Bull, and Y. Zheng. Content Extraction Signatures. In *Information Security and Cryptology — ICISC 2001*. Lecture Notes in Computer Science, vol 2288, pages 285–304

22. D. Stinson and M. Paterson. *Cryptography: Theory and Practice, Fourth Edition*. CRC Press, 4th edition, 2018.

23. G. M. Zaverucha and D. Stinson. Group testing and batch verification. In *Proceedings of the 4th International Conference on Information Theoretic Security, ICITS’09*, pages 140–157, 2009.