On the Security Notions for Homomorphic Signatures

Dario Catalano[1], Dario Fiore[2], and Luca Nizzardo[2]

[1] Dipartimento di Matematica e Informatica, Università di Catania, Italy.
catalano@dmi.unict.it
[2] IMDEA Software Institute, Madrid, Spain.
{dario.fiore, luca.nizzardo}@imdea.org

Abstract. Homomorphic signature schemes allow anyone to perform computation on signed data in such a way that the correctness of computation's results is publicly certified. In this work we analyze the security notions for this powerful primitive considered in previous work, with a special focus on adaptive security. Motivated by the complications of existing security models in the adaptive setting, we consider a simpler and (at the same time) stronger security definition inspired to that proposed by Gennaro and Wichs (ASIACRYPT'13) for homomorphic MACs. In addition to strength and simplicity, this definition has the advantage to enable the adoption of homomorphic signatures in dynamic data outsourcing scenarios, such as delegation of computation on data streams. Then, since no existing homomorphic signature satisfies this stronger notion, our main technical contribution are general compilers which turn a homomorphic signature scheme secure under a weak definition into one secure under the new stronger notion. Our compilers are totally generic with respect to the underlying scheme. Moreover, they preserve two important properties of homomorphic signatures: context-hiding (i.e. signatures on computation's output do not reveal information about the input) and efficient verification (i.e. verifying a signature against a program $\mathcal{P}$ can be made faster, in an amortized, asymptotic sense, than recomputing $\mathcal{P}$ from scratch).

1 Introduction

Digital signatures are a fundamental cryptographic primitive for guaranteeing the authenticity of digital information. In a digital signature scheme, a user Alice can use her secret key sk to generate a signature σ_m on a message m, and a user Bob can use Alice's public key pk to check the authenticity of (m, σ_m). The standard security notion of digital signatures, unforgeability against chosen message attacks, says that an attacker who has access to a collection of signatures on messages of his choice cannot produce a signature on a new message. This notion essentially means that signatures must be *non-malleable* in the sense that, from a signature on m one cannot derive a signature on some $m' \neq m$.

Even if in the most popular applications one wishes such a strong notion of non-malleability, there are application scenarios where some form of malleability can become very useful, paradoxically even for signature schemes. A notable example is that of homomorphic signatures, a notion first proposed by Desmedt [16] and Johnson et al. [24], and later properly formalized by Boneh and Freeman [6]. This is what we study in this work.

Homomorphic Signatures. In homomorphic signatures, a user Alice can use her secret key sk to generate signatures $\sigma_1, \ldots, \sigma_n$ on a collection of messages $(m_1, \ldots, m_n)$ – a so-called dataset. Then the interesting feature of this primitive is a (publicly computable) Eval algorithm that takes the signatures $\sigma_1, \ldots, \sigma_n$ and a program $\mathcal{P}$, and outputs a signature $\sigma_{\mathcal{P},m}$ on the message $m = \mathcal{P}(m_1, \ldots, m_n)$ *as the output of* $\mathcal{P}$. It is crucial that $\sigma_{\mathcal{P},m}$ is not a signature on just m, but on m

as output of the program $\mathcal{P}$. The latter observation indeed makes sure that signatures are not "too malleable", but they rather have a controlled malleability. This means that a user Bob will use Alice's public key pk to check the triple $(\mathcal{P}, m, \sigma_{\mathcal{P},m})$ and get convinced of whether m is the corret output of $\mathcal{P}$ on messages previously signed by Alice.

In addition to this interesting functionality, what makes this primitive attractive is the following set of features. First, homomorphic signatures must be *succinct*, meaning that their size must be significantly smaller than the size of the input dataset.[3] Second, Bob can verify computation's outputs without needing to know the original dataset, a very appealing feature when considering computations on very large datasets that could not be stored locally by verifiers. Third, homomorphic signatures are *composable*, in the sense that signatures obtained from Eval can be fed as inputs to new computations. Using composability, one can, for example, distribute different subtasks to several untrusted workers, ask each of them to produce a proof of its local task, and use these proofs to create another unique proof for the final job (as in the MapReduce approach). All these features make homomorphic signatures an interesting candidate to be used for securely delegating computation on previously outsourced data.

If the functionality of homomorphic signatures can be explained as above, defining the security notion of this primitive is a more delicate task. The following paragraphs provide an explanation of the security notions and then give an overview of our results. We warn the reader that the explanations in the introduction intentionally hide some details of the model for ease of exposition. A detailed formalization appears in Section 3.

Security of Homomorphic Signatures. Properly defining security for homomorphic signatures is tricky. Clearly, an homomorphic signature cannot meet the usual unforgeability requirement [22] as the primitive does allow the adversary to come up (honestly) with new signatures. The first satisfactory security definition was proposed by Boneh and Freeman in [6]. Intuitively, a homomorphic signature is secure if an adversary who knows the public key can only come up with signatures that are either obtained from the legitimate signer Alice, or they are obtained by running Eval on the signatures obtained by Alice. In other words, the adversary can only do what is in the scope of the public evaluation algorithm. Slightly more in detail, this new unforgeability game can be explained as follows. During a training phase the adversary $\mathcal{A}$ is allowed to see the signatures of messages belonging to different datasets. The adversary then wins the game if she can produce either (1) a signature on a message m belonging to some previously unseen dataset (this is called a *Type 1* forgery), or (2) for some previously seen dataset $\Delta = \{m_1, \ldots, m_n\}$, she manages to produce a triplet $(\mathcal{P}, \sigma, m)$, such that σ verifies correctly but $m \neq \mathcal{P}(m_1, \ldots, m_n)$ (this is called a *Type 2* forgery). Again explained in words, this definition means that the adversary can cheat either by claiming an output on a dataset that she never saw/queried, or by claiming an incorrect output of a given program $\mathcal{P}$, executed on a collection of messages for which she saw signatures.

A noteworthy caveat of the Boneh and Freeman [6] definition is the requirement that the adversary submits *all* the messages belonging to each queried dataset. Namely, for each queried dataset Δ, $\mathcal{A}$ has to ask *exactly* n signing queries.[4] In this work, because of this limitation, we call this notion *semi-adaptive security*.

[3] Without the succinctness requirement homomorphic signatures are trivial to realize as one can simply set $\sigma = (\mathcal{P}, (m_1, \sigma_1), \ldots, (m_\ell, \sigma_\ell))$.

[4] We remark that the original Boneh-Freeman definition imposes the even stronger restriction that these n messages are queried all at once.

To overcome this limitation, Freeman [18] later proposed a stronger notion where the adversary is allowed to adaptively query messages one by one, and even to sprinkle queries from different datasets. In this work, because of its increased adaptivity, we call the notion in [18] *adaptive security.*

The shortcomings of adaptive security. Adaptive security, while very natural, has a dark side. Loosening the query-all requirement implies that the adversary might provide a forgery $(\mathcal{P}, \sigma, m)$ that corresponds to a previously seen dataset Δ, but for which $\mathcal{A}$ did not ask signing queries on all the inputs of $\mathcal{P}$. For instance, $\mathcal{A}$ might pretend to have a signature on $m \neq \mathcal{P}(m_1, m_2)$ without having ever made a query on m_2. The issue in this case is that it is not even possible to define what is the correct output of $\mathcal{P}$ in order to say whether the adversary has cheated (i.e., if m is a correct output or not). To deal with this issue, Freeman proposed a notion of "well-defined program" which characterizes when the output of $\mathcal{P}$ can be defined in spite of missing inputs. The idea is simple and intuitively says that a program is well defined if the missing inputs do not change its outcome (e.g., $\mathcal{P}(m_1, \cdot)$ is constant). Freeman's definition then considered a forgery also one that passes verification for a $\mathcal{P}$ not well-defined, and called such a forgery *Type 3.*

Type 3 forgeries are however nasty animals. Not only they are very hard to work with (as the security definition turns complicated), but they also make the outcome of the security experiment not efficiently computable. In fact, when considering general functions it may not be possible to check the well-definedness of $\mathcal{P}$ in polynomial time. This can be solved when $\mathcal{P}$ is a linear [18] or a low degree polynomial [8,14], but the issue remains for the more general case, e.g., polynomial size circuits. In particular, this issue can generate troubles when proving the security of homomorphic signatures as well as when using them in larger protocols (as simply testing whether an adversary returned a forgery may not be doable in polynomial time).

1.1 Our contribution

The state of the art of security notions for homomorphic signatures, as discussed above, seems quite unsatisfactory. Having expressive, yet easy to use, definitions is indeed a fundamental step towards a better understanding of cryptographic primitives.

A Stronger and Simpler Security Notion. To address the issues of adaptive security, we consider a new security notion that is both simpler and stronger than the one in [18]. This notion, that we call *strong adaptive security*, is the public key version of the one proposed by Gennaro and Wichs [21] for homomorphic message authenticators (the secret key equivalent of homomorphic signatures).[5] Strong adaptive security deals with the case of programs with missing inputs in a simple way: if the triplet $(\mathcal{P}, m, \sigma)$ returned by the adversary verifies correctly and some inputs of $\mathcal{P}$ were not queried during the experiment, then it is considered a forgery (we call it a *Type 3 Strong* forgery).

Compared to previous notions, strong adaptive security has several advantages. First, the winning condition of the experiment is efficiently computable, thus avoiding the issues that may arise when proving and using homomorphic signatures. Second, the new forgery definition is arguably much simpler to state and work with. Finally, being a strengthening of adaptive security, homomorphic signature schemes that are strong adaptive secure can be used in more application scenarios as discussed before.

[5] With some adaptations to deal with multiple datasets which was not considered in [21].

Realizing Strong Adaptive Security, Generically. If we aim for strong adaptive security to be the "right" strong notion to use for homomorphic signatures, then we face the problem that virtually all existing schemes are not secure under this strong notion. This is the case for those schemes that support linear or low-degree polynomials and were proven secure under the adaptive notion of [18], as well as for the recently proposed leveled homomorphic scheme for circuits [23] which is only semi-adaptive secure. Notably, all these constructions break down in the new security experiment as they do not tolerate adversaries that issue Type 3 Strong forgeries. The only scheme which stands security in this stronger model is a recent proposal of Elkhiyaoui et al. [17] which supports constant-degree polynomials and relies on multilinear maps in the random oracle model.

To remedy this situation, our main contribution is to show that strong adaptive security can be easily achieved without additional assumptions and in the standard model. Specifically, our main result is a generic compiler that, starting from an homomorphic signature scheme Σ satisfying semi-adaptive security, converts Σ into a strong adaptive secure scheme that supports the same class of functions.

The compiler uses, as additional building block, a semi-adaptive secure signature scheme Σ_{OR} that supports OR operations over $\mathbb{Z}_2$. Clearly, if Σ supports arbitrary boolean circuits, then Σ_{OR} can be instantiated using Σ itself. In such a case, our result is thus providing a transformation that "bootstraps" semi-adaptive security to strong adaptive security. If, on the other hand, Σ_{OR} cannot be instantiated using Σ, our result still provides a way to get strong adaptive security, under the additional assumption that semi-adaptive secure OR-homomorphic signatures exist.

Nevertheless, since very few concrete examples of OR-homomorphic signatures are known (essentially, only one [23]) we asked whether a similar result could be obtained out of some more widely studied primitives. Along this direction, our second result is another compiler that combines a semi-adaptive secure scheme Σ together with a semi-adaptive secure *linearly-homomophic* signature Σ_{LH} that works for messages over a large ring, say $\mathbb{Z}_p$. This combination yields a homomorphic signature scheme that is strong adaptive secure and supports the same class of functions supported by Σ. A limitation of this second transformation is that it applies only to schemes that are leveled homomorphic (i.e., for circuits of bounded depth). As an interesting feature, however, this result shows that strong adaptive security can be obtained from linearly-homomorphic schemes, a class of constructions for which many constructions are known (most of which are also way more efficient in practice than [23]).

Both our transformations hold in the standard model, and they preserve two properties of homomorphic signatures: context-hiding and efficient-verification (so, security can be upgraded without penalties). The former deals with privacy and informally says that signatures on computation's outputs do not reveal information on the inputs. The latter instead fully enables the use of homomorphic signatures for verifiable delegation of computation, by requiring that verifying a signature for a program $\mathcal{P}$ is asymptotically faster (in an amortized, offline-online sense) than recomputing $\mathcal{P}$ from scratch.

We point out that our compilers are completely generic with respect to the semi adaptive secure scheme. This means, for instance, that when applied to the recent (leveled) fully homomorphic solution of [23] they lead to homomorphic signature schemes for general circuits achieving strong adaptive security.

On the Importance of Strong Adaptive Security. As an important application of (strong) adaptive secure homomorphic signatures, we mention *certified computation on streaming data.* Consider a scenario where a user Alice outsources a stream of data $m_1, m_2, \ldots$ to an untrusted Cloud, so that the Cloud can compute a program $\mathcal{P}$ on the current snapshot $(m_1, \ldots, m_i)$ and post the result publicly (e.g., on a third party website). Using homomorphic signatures, Alice can sign each element of the data stream, while the Cloud can compute a homomorphic signature $\sigma_{\mathcal{P},y_i}$ on every computed result $y_i = \mathcal{P}(m_1, \ldots, m_i)$ and post $(y_i, \sigma_{\mathcal{P},y_i})$. This way, anyone with the only knowledge of Alice's public key is able to check the results validity. Notably, the Cloud can produce the certified results in a completely non-interactive fashion, and no communication between Alice and the verifiers is needed (except, of course, for sending the public key). In such a scenario, where datasets grow dynamically and one performs computations on their current version, (strong) adaptive security is fundamental as it prevents the cloud from claiming to have results computed on dataset elements that it did not receive (yet). This is particularly relevant in scenarios where there is no communication between the signer and the verifiers, who may not be aware of the current status of the outsourced stream. Furthermore, strong adaptive security is important in the case of very large, potentially unbounded, datasets (as in the streaming case) as one cannot assume that the adversary queries the whole dataset. This actually shows an inherent limitation of semi-adaptive security: this latter model cannot cope with dataset of arbitrarily large, possibly exponential, size. Indeed, to fit the requirements of the definition, polynomially bounded adversaries would be required to ask an equal (yet impossibly large) number of signing queries.

1.2 Other related work

The notion of homomorphic signature was (informally) suggested by Desmedt [16] and later more formally introduced by Johnson *et al.* [24]. The special case of linearly homomorphic signatures was first considered by Boneh *et al.* [5] as a key tool to prevent pollution attacks in network coding routing mechanisms. Following this work, several papers further studied this primitive both in the random oracle [20,7,6,10], and in the standard model [2,12,3,13,18,4,11]. In the symmetric setting realizations of linearly homomorphic MACs have been proposed by Agrawal *et al.* in [1].

Several recent works also considered the question of constructing homomorphic authenticators (i.e., signatures and/or MACs) supporting more expressive functionalities. Boneh and Freeman in [6] proposed an homomorphic signature scheme for constant degree polynomials, in the random oracle model. Gennaro and Wichs [21] presented a construction of fully homomorphic MACs based on fully homomorphic encryption in a restricted adversarial model where no verification queries are allowed. Catalano and Fiore [8] proposed a much more efficient homomorphic MAC solution that, while capturing a less expressive class of functionalities (i.e. arithmetic circuits of polynomially bounded degree), allows for verification queries. This latter result was further generalized in [9]. All these constructions of homomorphic MACs achieve adaptive security.

In the asymmetric setting, Catalano, Fiore and Warinschi [14] proposed a homomorphic signature that achieves adaptive security in the standard model, works for constant degree polynomials and is based on multilinear maps. Moreover, Gorbunov, Vaikuntanathan and Wichs [23] recently proposed the first homomorphic signature construction that can handle boolean circuits of bounded polynomial depth; their scheme is secure in the semi-adaptive model, and is based on standard lattices.

Finally, we notice that a work from Chase et al. [15] considered a problem similar to the one addressed in this work (i.e., elaborating a definition that allows one to establish, in an efficient way, when the signature produced by the adversary is a valid forgery). They deal with this problem by formalizing the idea that the adversary "must know" the function and the input that were used to obtain the forgery. To formalize this idea, their definition asks for the existence of a black-box extractor that must extract this information from what is in the view of the game and the output of the adversary. Unfortunately, this type of definition is impossible to achieve when one considers the case of *succinct* homomorphic signatures for n-ary functions, as we do in our paper. The reason is simply that the extractor should extract an amount of information (such as the function input) that is much larger than what is in its input.

2 Preliminaries

Notation. We denote with $\lambda \in \mathbb{N}$ a security parameter. A *probabilistic polynomial time* (PPT) algorithm $\mathcal{A}$ is a randomized algorithm for which there exists a polynomial $p(\cdot)$ such that for every input x the running time of $\mathcal{A}(x)$ is bounded by $p(|x|)$. We say that a function $\epsilon : \mathbb{N} \to \mathbb{R}^+$ is *negligible* if for every positive polynomial $p(\lambda)$ there exists $\lambda_0 \in \mathbb{N}$ such that for all $\lambda > \lambda_0$: $\epsilon(\lambda) < 1/p(\lambda)$. If S is a set, $x \overset{\$}{\leftarrow} S$ denotes the process of selecting x uniformly at random in S. If $\mathcal{A}$ is a probabilistic algorithm, $y \overset{\$}{\leftarrow} \mathcal{A}(\cdot)$ denotes the process of running $\mathcal{A}$ on some appropriate input and assigning its output to y. For a positive integer n, we denote by $[n]$ the set $\{1, \ldots, n\}$.

3 Homomorphic Signatures

In this section we recall the definition of homomorphic signatures. This definition extends the one by Freeman in [18] in order to work with the general notion of labeled programs [21].

Labeled Programs [21]. A *labeled program* $\mathcal{P}$ is a tuple $(f, \tau_1, ..., \tau_n)$ such that $f : \mathcal{M}^n \to \mathcal{M}$ is a function of n variables (e.g., a circuit) and $\tau_i \in \{0,1\}^*$ is a label of the i-th input of f. Labeled programs can be composed as follows: given $\mathcal{P}_1, \ldots, \mathcal{P}_t$ and a function $g : \mathcal{M}^t \to \mathcal{M}$, the composed program $\mathcal{P}^*$ is the one obtained by evaluating g on the outputs of $\mathcal{P}_1, \ldots, \mathcal{P}_t$, and it is denoted as $\mathcal{P}^* = g(\mathcal{P}_1, \ldots, \mathcal{P}_t)$. The labeled inputs of $\mathcal{P}^*$ are all the distinct labeled inputs of $\mathcal{P}_1, \ldots, \mathcal{P}_t$ (all the inputs with the same label are grouped together and considered as a unique input of $\mathcal{P}^*$).
Let $f_{id} : \mathcal{M} \to \mathcal{M}$ be the identity function and $\tau \in \{0,1\}^*$ be any label. We refer to $\mathcal{I}_\tau = (f_{id}, \tau)$ as the identity program with label τ. Note that a program $\mathcal{P} = (f, \tau_1, \cdots, \tau_n)$ can be expressed as the composition of n identity programs $\mathcal{P} = f(\mathcal{I}_{\tau_1}, \cdots, \mathcal{I}_{\tau_n})$.

Definition 1 (Homomorphic Signature). *A homomorphic signature scheme* HSig *consists of a tuple of PPT algorithms* $(\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Ver}, \mathsf{Eval})$ *with the following syntax:*

$\mathsf{KeyGen}(1^\lambda, \mathcal{L})$ *the key generation algorithm takes as input a security parameter λ, a description of the label space $\mathcal{L}$ (which fixes the maximum data set size N), and outputs a public key* vk *and a secret key* sk*. The public key* vk *contains a description of the message space $\mathcal{M}$ and the set $\mathcal{F}$ of admissible functions.*

$\mathsf{Sign}(\mathsf{sk}, \Delta, \tau, m)$ *the signing algorithm takes as input a secret key* sk*, a data set identifier $\Delta \in \{0,1\}^*$, a label $\tau \in \mathcal{L}$, a message $m \in \mathcal{M}$, and it outputs a signature σ.*

$\mathsf{Eval}(\mathsf{vk}, f, \sigma_1, \ldots, \sigma_n)$ *the evaluation algorithm takes as input a public key* vk*, a function $f \in \mathcal{F}$ and a tuple of signatures $\{\sigma_i\}_{i=1}^n$ (assuming that f takes n inputs). It outputs a new signature σ.*

$\mathsf{Ver}(\mathsf{vk}, \mathcal{P}, \Delta, m, \sigma)$ *the verification algorithm takes as input a public key* vk*, a labeled program $\mathcal{P} = (f, \tau_1, \ldots, \tau_n)$ with $f \in \mathcal{F}$, a dataset identifier Δ, a message $m \in \mathcal{M}$, and a signature σ. It outputs either 0 (reject) or 1 (accept).*

A homomorphic signature scheme is required to satisfy the properties of *authentication correctness*, *evaluation correctness* and *succinctness* that we describe below. The security property is discussed slightly later in Section 3.1.

Authentication Correctness. Intuitively, a homomorphic signature scheme has authentication correctness if the signature generated by $\mathsf{Sign}(\mathsf{sk}, \varDelta, \tau, m)$ verifies correctly for m as the output of the identity program $\mathcal{I}_\tau$ on a dataset with identifier $\varDelta$. More formally, a scheme HSig satisfies the authentication correctness property if for a given label space $\mathcal{L}$, all key pairs $(\mathsf{sk}, \mathsf{vk}) \leftarrow \mathsf{KeyGen}(1^\lambda, \mathcal{L})$, any label $\tau \in \mathcal{L}$, dataset identifier $\varDelta \in \{0,1\}^*$, and any signature $\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}, \varDelta, \tau, m)$, $\mathsf{Ver}(\mathsf{vk}, \mathcal{I}_\tau, \varDelta, m, \sigma)$ outputs 1 with all but negligible probability.

Evaluation Correctness. Intuitively, this property says that running the evaluation algorithm on signatures $(\sigma_1, \ldots, \sigma_t)$ such that each σ_i verifies for m_i as the output of a labeled program $\mathcal{P}_i$ and a dataset with identifier $\varDelta$, produces a signature σ which verifies for $g(m_1, \ldots, m_t)$ as the output of the composed program $g(\mathcal{P}_1, \ldots, \mathcal{P}_t)$ and same dataset $\varDelta$. More formally, fix a key pair $(\mathsf{vk}, \mathsf{sk}) \overset{\$}{\leftarrow} \mathsf{KeyGen}(1^\lambda, \mathcal{L})$, a function $g : \mathcal{M}^t \rightarrow \mathcal{M}$, and any set of program/message/signature triples $\{(\mathcal{P}_i, m_i, \sigma_i)\}_{i=1}^t$ such that $\mathsf{Ver}(\mathsf{vk}, \mathcal{P}_i, \varDelta, m_i, \sigma_i) = 1$. If $m^* = g(m_1, \ldots, m_t)$, $\mathcal{P}^* = g(\mathcal{P}_1, \ldots, \mathcal{P}_t)$, and $\sigma^* = \mathsf{Eval}(\mathsf{vk}, g, \sigma_1, \ldots, \sigma_t)$, then $\mathsf{Ver}(\mathsf{vk}, \mathcal{P}^*, \varDelta, m^*, \sigma^*) = 1$ holds with all but negligible probability.

Succinctness. A homomorphic signature scheme is said to be *succinct* if, for a fixed security parameter λ, the size of signatures depends at most logarithmically on the size of the input dataset. More formally, HSig satisfies succinctness if there exists a polynomial $p(\lambda)$ such that for all $(\mathsf{vk}, \mathsf{sk}) \overset{\$}{\leftarrow} \mathsf{KeyGen}(1^\lambda, \mathcal{L})$, all $(m_1, \ldots, m_t) \in \mathcal{M}^t$, all $(\tau_1, \ldots, \tau_t) \in \mathcal{L}^t$, any $\varDelta \in \{0,1\}^*$, and all functions $f \in \mathcal{F}$, if $\sigma_i \overset{\$}{\leftarrow} \mathsf{Sign}(\mathsf{sk}, \varDelta, \tau_i, m_i)$ and $\sigma \leftarrow \mathsf{Eval}(\mathsf{vk}, f, \sigma_1, \ldots, \sigma_t)$, then $|\sigma| \leq p(\lambda) \cdot \log t$.

3.1 Security

At an intuitive level, a homomorphic signature is secure if an adversary, without knowledge of the secret key, can only come up with signatures that it obtained from the signer, or signatures that are obtained by running the Eval algorithm on signatures obtained from the legitimate signer. Formalizing this intuition turns out to be tricky and leaves space to different possibilities.

In what follows we present three different security notions for homomorphic signatures that we call *semi-adaptive*, *adaptive*, and *strong adaptive*, respectively. These notions share the same security experiment between an adversary $\mathcal{A}$ and a challenger, and the only difference lies in what is considered a forgery. The security experiment, denoted $\mathbf{Exp}^{\mathsf{UF}}_{\mathcal{A},\mathsf{HSig}}(\lambda)$, proceeds as described below:

Key Generation The challenger runs $(\mathsf{vk}, \mathsf{sk}) \overset{\$}{\leftarrow} \mathsf{KeyGen}(1^\lambda, \mathcal{L})$ and gives vk to $\mathcal{A}$.

Signing Queries $\mathcal{A}$ can adaptively submit queries of the form $(\varDelta, \tau, m)$, where $\varDelta$ is a data set identifier, $\tau \in \mathcal{L}$, and $m \in \mathcal{M}$. The challenger proceeds as follows:
- if $(\varDelta, \tau, m)$ is the first query with the data set identifier $\varDelta$, the challenger initializes an empty list $T_\varDelta = \emptyset$ for $\varDelta$.
- If $T_\varDelta$ does not already contain a tuple $(\tau, \cdot)$ (i.e., $\mathcal{A}$ never asked for a query $(\varDelta, \tau, \cdot)$), the challenger computes $\sigma \overset{\$}{\leftarrow} \mathsf{Sign}(\mathsf{sk}, \varDelta, \tau, m)$, returns σ to $\mathcal{A}$ and updates the list $T_\varDelta \leftarrow T_\varDelta \cup (\tau, m)$.
- If $(\tau, m) \in T_\varDelta$ (i.e., the adversary had already queried the tuple $(\varDelta, \tau, m)$), the challenger replies with the same signature generated before.

– If T_Δ contains a tuple (τ, m') for some message $m' \neq m$, then the challenger ignores the query. Note that this means that a tuple $(\Delta, \tau, \cdot)$ can be queried only once.

Forgery The previous stage is executed until the adversary $\mathcal{A}$ outputs a tuple $(\mathcal{P}^*, \Delta^*, m^*, \sigma^*)$. The experiments outputs 1 if the tuple returned by $\mathcal{A}$ is a forgery, and 0 otherwise.

To complete the description of the experiment, it remains to define when a tuple $(\mathcal{P}^*, \Delta^*, m^*, \sigma^*)$ is considered a forgery. We give below three different forgery definitions; each of them yields a corresponding security notion for the homomorphic signature scheme.

Semi-Adaptive Secure Homomorphic Signatures. Informally speaking, in the semi-adaptive security game a forgery is one where either (1) the dataset Δ^* is "new" (i.e., no signing query $(\Delta^*, \cdot, \cdot)$ was ever made during the game), or (2) the claimed output m^* of $\mathcal{P}^*$ is not the correct one. The crucial aspect of this definition is that to identify what is a correct output, one assumes that the adversary has fully specified the inputs of $\mathcal{P}^*$, namely $\mathcal{A}$ has asked for signatures on $(\Delta^*, \tau_i^*, m_i)$, for all $i = 1$ to n. More formally,

Definition 2 (Semi-Adaptive Security). *We define* $\mathbf{Exp}^{\mathsf{semi\text{-}Ad\text{-}UF}}_{\mathcal{A},\mathsf{HSig}}(\lambda)$ *as the security experiment which proceeds as* $\mathbf{Exp}^{\mathsf{UF}}_{\mathcal{A},\mathsf{HSig}}(\lambda)$ *with the addition that the tuple* $(\mathcal{P}^* := (f^*, \tau_1^*, \ldots, \tau_n^*), \Delta^*, m^*, \sigma^*)$ *returned by the adversary* $\mathcal{A}$ *is considered a forgery if* $\mathsf{Ver}(\mathsf{vk}, \mathcal{P}^*, \Delta^*, m^*, \sigma^*) = 1$ *and either one of the following conditions hold:*

Type 1: *The list* T_{Δ^*} *has not been initialised during the game.*
Type 2: *For all* $i \in [n]$, $\exists\, (\tau_i, m_i) \in T_{\Delta^*}$ *and* $m^* \neq f^*(m_1, \ldots, m_n)$.

Let $\mathbf{Adv}^{\mathsf{semi\text{-}Ad\text{-}UF}}_{\mathcal{A},\mathsf{HSig}}(\lambda) = \Pr[\mathbf{Exp}^{\mathsf{semi\text{-}Ad\text{-}UF}}_{\mathcal{A},\mathsf{HSig}}(\lambda) = 1]$ *be the advantage of* $\mathcal{A}$ *against the semi-adaptive security of scheme* HSig*. We say that a homomorphic signature scheme* HSig *is* semi-adaptive secure *(or simply secure) if for every PPT adversary* $\mathcal{A}$ *there exists a negligible function* $\epsilon(\lambda)$ *such that* $\mathbf{Adv}^{\mathsf{semi\text{-}Ad\text{-}UF}}_{\mathcal{A},\mathsf{HSig}}(\lambda) \leq \epsilon(\lambda)$.

We stress that in the above security experiment the adversary $\mathcal{A}$ is restricted to produce Type 2 forgeries where *all* the inputs of the labeled program have been queried during the experiment. This notion works well for applications where the dataset is signed in one shot (as in the earlier proposals of homomorphic signatures [6]), or where one computes on the signed data only *after* the whole dataset has been filled up. In contrast, in those applications where the dataset is signed incrementally and one performs computations in between (e.g., in streaming applications), semi-adaptive security falls short of providing good guarantees. The issue is that in such a dynamic setting the adversary may claim a forgery with a labeled program containing a label τ^* that was not queried during the game. In this case, the input of $\mathcal{P}^*$ is no longer specified and defining whether the adversary's output is a forgery is not captured by Definition 2.

From the literature, we note that the schemes in [5,20,6,23] are proven under a weaker version of semi-adaptive security where the messages of every dataset have to be queried all at once.[6]

Adaptive Secure Homomorphic Signatures. The issue of adversaries who claim programs in which some of the inputs are missing in the forgery stage was recognized earlier on by Freeman [18]. To deal with this issue, he proposed a notion of "well-defined programs" which characterizes when the output of a program can be defined in spite of missing inputs. The idea is rather simple and intuitively says that a program is well-defined if the missing inputs do not change its outcome.

[6] Actually, the authors of [23] mention that the proof of their scheme can be modified to hold under a definition with adaptive queries to data items, corresponding to the semi-adaptive security presented in this paper.

Definition 3 (Well-Defined Labeled Program [18]). *A labeled program $\mathcal{P}^* = (f^*, \tau_1^*, \ldots, \tau_n^*)$ is well-defined with respect to a list $T = \{(\tau_i, m_i)\}_i$ if one of the two following cases holds:*

- $\forall i = 1, \ldots, n : (\tau_i^*, m_i) \in T$.
- $\exists\ j \in \{1, \cdots, n\}$ *s.t.* $(\tau_j, \cdot) \notin T$, *and for all possible choices of* $\tilde{m}_j \in \mathcal{M}$ *such that* $(\tau_j, \cdot) \notin T$ $f^*(m'_1, \ldots, m'_n)$ *is the same, where* $m'_i = m_i$ *for all* i *s.t.* $(\tau_i, m_i) \in T$ *and* $m'_i = \tilde{m}_i$ *otherwise.*

With the above notion of well-defined programs, adaptive security can be defined as follows.

Definition 4 (Adaptive Security [18]). *We define* $\mathbf{Exp}^{\mathsf{Ad\text{-}UF}}_{\mathcal{A},\mathsf{HSig}}(\lambda)$ *as the security experiment which proceeds as* $\mathbf{Exp}^{\mathsf{UF}}_{\mathcal{A},\mathsf{HSig}}(\lambda)$ *with the addition that the tuple* $(\mathcal{P}^* := (f^*, \tau_1^*, \ldots, \tau_n^*), \Delta^*, m^*, \sigma^*)$ *returned by the adversary* $\mathcal{A}$ *is considered a forgery if* $\mathsf{Ver}(\mathsf{vk}, \mathcal{P}^*, \Delta^*, m^*, \sigma^*) = 1$ *and either one of the following conditions hold:*

Type 1: *The list* T_{Δ^*} *has not been initialized during the game.*
Type 2: $\mathcal{P}^*$ *is well-defined with respect to* T_{Δ^*}, *and* $m^* \neq f^*(m'_1, \ldots, m'_n)$ *where* $m'_i = m_i$ *for all* i *s.t.* $(\tau_i, m_i) \in T_{\Delta^*}$ *and* $m'_i = \tilde{m}$ *(for some arbitrary* $\tilde{m} \in \mathcal{M}$*), otherwise.*
Type 3: $\mathcal{P}^*$ *is* not *well-defined with respect to* T_{Δ^*}.

Let $\mathbf{Adv}^{\mathsf{Ad\text{-}UF}}_{\mathcal{A},\mathsf{HSig}}(\lambda) = \Pr[\mathbf{Exp}^{\mathsf{Ad\text{-}UF}}_{\mathcal{A},\mathsf{HSig}}(\lambda) = 1]$ *be the advantage of* $\mathcal{A}$ *against the adaptive security of scheme* HSig*. We say that a homomorphic signature scheme* HSig *is* adaptive secure *if for every PPT adversary* $\mathcal{A}$ *there exists a negligible function* $\epsilon(\lambda)$ *such that* $\mathbf{Adv}^{\mathsf{Ad\text{-}UF}}_{\mathcal{A},\mathsf{HSig}}(\lambda) \leq \epsilon(\lambda)$.

Comparing the above definition of adaptive security with the semi-adaptive definition presented earlier, we note the following: Type 1 forgeries are identical in both definitions. Type 2 forgeries are similar: intuitively, they both capture the case when the adversary cheats on the result of $\mathcal{P}^*$, except that Definition 4 addresses the case of missing inputs by defining what is, in this case, a correct output (using the notion of well-defined program). Finally, Type 3 forgeries are introduced in Definition 4 to address the remaining case in which $\mathcal{P}^*$ may have different outputs, yet the forgery verifies correctly.

From the literature, the schemes in [18,2,12,13,10,14,11] are proven under the adaptive security notion presented above.

Strong Adaptive Secure Homomorphic Signatures. The good of the adaptive definition given above is that it addresses the issue of labeled programs with unspecified inputs by modeling when an adversary is cheating. The modeling of Definition 4 however comes at the price of a rather cumbersome security definition. Well-defined programs are certainly not the most intuitive notion to work with. In addition, besides simplicity, the main issue with the above notion is that deciding whether the tuple returned by the adversary is a forgery may not be doable in polynomial time. Indeed, making this test would require to execute f^* on all possible values of the missing inputs (that may be exponentially many). In the case when admissible functions are low-degree arithmetic circuits over a large field, it has been shown that well-defined programs can be tested probabilistically, and that Type 3 forgeries can be reduced to Type 2 ones [9]. However, for general circuits the inefficient test issue remains and can generate troubles when proving the security of homomorphic signature schemes as well as when using them in larger protocols (as simply testing whether an adversary returned a forgery – wins – may not be doable in polynomial time).

To address this issue, in what follows we consider a stronger and much simpler security definition. This notion is obtained by extending the notion of semi-adaptive security (Definition 2) with a very simple notion of Type 3 forgeries. The latter are just forgeries where the labeled program contains a "new" label. The formal definition follows.

Definition 5 (Strong Adaptive Security). *We define* $\mathbf{Exp}^{\mathsf{strong\text{-}Ad\text{-}UF}}_{\mathcal{A},\mathsf{HSig}}(\lambda)$ *as the security experiment which proceeds as* $\mathbf{Exp}^{\mathsf{UF}}_{\mathcal{A},\mathsf{HSig}}(\lambda)$ *except that the tuple* $(\mathcal{P}^* := (f^*, \tau_1^*, \ldots, \tau_n^*), \Delta^*, m^*, \sigma^*)$ *returned by the adversary* $\mathcal{A}$ *is considered a forgery if* $\mathsf{Ver}(\mathsf{vk}, \mathcal{P}^*_{\Delta^*}, m^*, \sigma^*) = 1$ *and either one of the following conditions hold:*

Type 1: *The list* T_{Δ^*} *has not been initialized during the game.*
Type 2: *For all* $i \in [n]$, $\exists\, (\tau_i, m_i) \in T_{\Delta^*}$ *and* $m^* \neq f^*(m_1, \ldots, m_n)$.
Type 3 Strong: *there exists* $j \in \{1, \ldots, n\}$ *such that* $(\tau_j^*, \cdot) \notin T_{\Delta^*}$.

Let $\mathbf{Adv}^{\mathsf{strong\text{-}Ad\text{-}UF}}_{\mathcal{A},\mathsf{HSig}}(\lambda) = \Pr[\mathbf{Exp}^{\mathsf{strong\text{-}Ad\text{-}UF}}_{\mathcal{A},\mathsf{HSig}}(\lambda) = 1]$ *be the advantage of* $\mathcal{A}$ *against the strong adaptive security of scheme* HSig. *We say that a homomorphic signature scheme* HSig *is* strong adaptive secure *if for every PPT adversary* $\mathcal{A}$ *there exists a negligible function* $\epsilon(\lambda)$ *such that* $\mathbf{Adv}^{\mathsf{strong\text{-}Ad\text{-}UF}}_{\mathcal{A},\mathsf{HSig}}(\lambda) \leq \epsilon(\lambda)$.

It is easy to see that the security notion of Definition 5 now allows to detect forgeries in polynomial time, and is without doubt much simpler than Definition 4. Basically, this notion is the public-key equivalent of the security notion proposed by Gennaro and Wichs [21] for fully-homomorphic MACs (with some cosmetic changes due to the handling of multiple datasets).

Relation between security notions. We note that the three security definitions presented in this Section are increasingly strong. Definition 4 is strictly stronger than Definition 2: while all forgeries in $\mathbf{Exp}^{\mathsf{semi\text{-}Ad\text{-}UF}}_{\mathcal{A},\mathsf{HSig}}(\lambda)$ are also forgeries in $\mathbf{Exp}^{\mathsf{Ad\text{-}UF}}_{\mathcal{A},\mathsf{HSig}}(\lambda)$, the converse is not true as any forgery in $\mathbf{Exp}^{\mathsf{Ad\text{-}UF}}_{\mathcal{A},\mathsf{HSig}}(\lambda)$ where the labeled program $\mathcal{P}^*$ contains an unqueried label is not considered a forgery in $\mathbf{Exp}^{\mathsf{semi\text{-}Ad\text{-}UF}}_{\mathcal{A},\mathsf{HSig}}(\lambda)$.

Definition 5 is strictly stronger than Definition 4. In one direction, any Type 1 and Type 3 forgery in $\mathbf{Exp}^{\mathsf{Ad\text{-}UF}}_{\mathcal{A},\mathsf{HSig}}(\lambda)$ yields, respectively, a Type 1 and a Type 3 Strong forgery in $\mathbf{Exp}^{\mathsf{strong\text{-}Ad\text{-}UF}}_{\mathcal{A},\mathsf{HSig}}(\lambda)$, and a Type 2 forgery in $\mathbf{Exp}^{\mathsf{Ad\text{-}UF}}_{\mathcal{A},\mathsf{HSig}}(\lambda)$ becomes either a Type 2 forgery or a Type 3 Strong forgery in $\mathbf{Exp}^{\mathsf{strong\text{-}Ad\text{-}UF}}_{\mathcal{A},\mathsf{HSig}}(\lambda)$. In the other direction, there exist forgeries in experiment $\mathbf{Exp}^{\mathsf{strong\text{-}Ad\text{-}UF}}_{\mathcal{A},\mathsf{HSig}}(\lambda)$ that are not considered so in $\mathbf{Exp}^{\mathsf{Ad\text{-}UF}}_{\mathcal{A},\mathsf{HSig}}(\lambda)$. We show this by considering the following adversary $\mathcal{A}$. $\mathcal{A}$ asks signing queries $(\Delta, \tau_1, m_1), (\Delta, \tau_2, m_2)$ and obtains signatures σ_1, σ_2; it computes $\sigma^* \leftarrow \mathsf{Eval}(\mathsf{vk}, \times, \sigma_1, \sigma_2)$, and outputs $(\mathcal{P}^* := (f, \tau_1, \tau_2, \tau_3), \Delta, m_1 \cdot m_2)$, where f is the function $f(x, y, z) = x(y+z) - xz$.[7] As one can see, the output of $\mathcal{A}$ is a Type 3 Strong forgery, since τ_3 is a label which has never been queried, while it is not a forgery in $\mathbf{Exp}^{\mathsf{Ad\text{-}UF}}_{\mathcal{A},\mathsf{HSig}}(\lambda)$, since $\mathcal{P}^* := (f, \tau_1, \tau_2, \tau_3)$ is well-defined with respect to the set of queries $T_\Delta = \{(\tau_1, m_1), (\tau_2, m_2)\}$, and $m_1 \cdot m_2$ is the correct output.

In addition to the fact that the security notions are strictly separated, we also note that by using a counterexample such as the one above it is possible to show that previously proposed homomorphic signatures (e.g., [6,14,23]) are *not* strong adaptive secure.

3.2 Context Hiding

As an additional property, homomorphic signatures can be required to satisfy a privacy property called *context-hiding* [6]. Very intuitively, a homomorphic signature is context hiding if signatures on functions' outputs do not reveal information about the originally signed inputs. Here we recall

[7] Any other function where the third input cancels out would work. Furthermore, although in the given example it is trivial to recognize that $\mathcal{P}$ is well-defined, this may not be the case for general functions.

the definition of context-hiding as given in [11, full version]. The notion in [11] generalizes the one proposed by Gorbunov, Vaikuntanathan and Wichs [23] to the setting of labeled programs. We defer the interested reader to the discussion in [11] for details on the differences and its relation to other prior context-hiding definitions.

Definition 6 (Context-Hiding [11]). *An homomorphic signature scheme for labeled programs supports context-hiding if there exist additional PPT procedures* $\tilde{\sigma} \leftarrow \mathsf{Hide}(\mathsf{vk}, m, \sigma)$ *and* $\mathsf{HVerify}(\mathsf{vk}, \mathcal{P}, \Delta, m, \sigma)$ *such that:*

- *Correctness: For any* $(\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda, \mathcal{L})$ *and any tuple* $(\mathcal{P}, \Delta, m, \sigma)$ *such that* $\mathsf{Ver}(\mathsf{vk}, \mathcal{P}, \Delta, m, \sigma) = 1$*, we have that if* $\tilde{\sigma} \leftarrow \mathsf{Hide}(\mathsf{vk}, m, \sigma)$ *then* $\mathsf{HVerify}(\mathsf{vk}, \mathcal{P}, \Delta, m, \tilde{\sigma}) = 1$.
- *Unforgeability: The signature scheme is secure when we replace the original verification algoritm* Ver *with* $\mathsf{HVerify}$ *in the security game.*
- *Context-Hiding Security: There is a simulator* Sim *such that, for any fixed (worst-case) choice of* $(\mathsf{sk}, \mathsf{vk}) \in \mathsf{KeyGen}(1^\lambda, \mathcal{L})$*, any labeled program* $\mathcal{P} = (f, \tau_1, \ldots, \tau_\ell)$*, data-set* Δ *and messages* $m_1, \ldots, m_\ell$*, there exists a function* $\epsilon(\lambda)$ *such that for any distinguisher* $\mathcal{D}$ *it holds*

$$| \Pr[\mathcal{D}(I, \mathsf{Hide}(\mathsf{vk}, m, \sigma)) = 1] - \Pr[\mathcal{D}(I, \mathsf{Sim}(\mathsf{sk}, \mathcal{P}, \Delta, m) = 1] | = \epsilon(\lambda)$$

  where $I = (\mathsf{sk}, \mathsf{vk}, \mathcal{P}, \Delta, \{m_i, \sigma_i = \mathsf{Sign}(\mathsf{sk}, \Delta, \tau_i, m_i)\}_{i=1}^{\ell})$*,* $m = f(m_1, \ldots, m_\ell)$*,* $\sigma \leftarrow \mathsf{Eval}(\mathsf{vk}, \sigma_1, \ldots, \sigma_\ell)$*, and the probabilities are taken over the randomness of* $\mathsf{Sign}, \mathsf{Hide}$ *and* Sim*. If* $\epsilon(\lambda)$ *is negligible then the scheme is said to have* statistical *context-hiding, otherwise, if* $\epsilon(\lambda) = 0$*, the scheme has* perfect *context-hiding.*

3.3 Efficient Verification

We recall the notion of homomorphic signatures with efficient verification introduced in [14]. This property is interesting as it enables the use of homomorphic signatures for verifiable delegation of computation in the amortized model of [19]. The property states that the verification algorithm can be split in two phases: an *offline* phase where, given the verification key vk and a labeled program $\mathcal{P}$, one precomputes a concise key $\mathsf{vk}_\mathcal{P}$; an *online* phase in which $\mathsf{vk}_\mathcal{P}$ can be used to verify signatures w.r.t. $\mathcal{P}$ and *any* dataset Δ. To achieve (amortized) efficiency, the idea is that $\mathsf{vk}_\mathcal{P}$ can be reused an unbounded number of times, and the online verification is cheaper than running $\mathcal{P}$. More formally:

Definition 7 (Efficient Verification). *Let* $\mathsf{HSig} = (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Ver}, \mathsf{Eval})$ *be a homomorphic signature scheme for labeled programs.* HSig satisfies efficient verification *if there exist two additional algorithms* $(\mathsf{VerPrep}, \mathsf{EffVer})$ *such that:*

$\mathsf{VerPrep}(\mathsf{vk}, \mathcal{P})$**:** *given in input the verification key* vk *and a labeled program* $\mathcal{P} = (f, \tau_1, \ldots, \tau_n)$*, the verification preparation algorithm generates a concise verification key* $\mathsf{vk}_\mathcal{P}$.

$\mathsf{EffVer}(\mathsf{vk}_\mathcal{P}, \Delta, m, \sigma)$**:** *on input a verification key* $\mathsf{vk}_\mathcal{P}$*, a data set identifier* Δ*, a message* $m \in \mathcal{M}$ *and a signature* σ*, the efficient verification algorithm outputs 0 (reject) or 1 (accept).*

The above algorithms are required to satisfy the following two properties:

Correctness. *Let* $(\mathsf{sk}, \mathsf{vk}) \xleftarrow{\$} \mathsf{KeyGen}(1^\lambda, \mathcal{L})$ *be honestly generated keys, and* $(\mathcal{P}, \Delta, m, \sigma)$ *be any program/dataset/message/signature tuple with* $\mathcal{P}$ *such that* $\mathsf{Ver}(\mathsf{vk}, \mathcal{P}, \Delta, m, \sigma) = 1$*. Then, for every* $\mathsf{vk}_\mathcal{P} \xleftarrow{\$} \mathsf{VerPrep}(\mathsf{vk}, \mathcal{P})$*,* $\mathsf{EffVer}(\mathsf{vk}_\mathcal{P}, \Delta, m, \sigma) = 1$ *holds with all but negligible probability.*

Amortized Efficiency. *Let* $\mathcal{P} = (f, \tau_1, \ldots, \tau_n)$ *be a labeled program and* Δ *be a dataset identifier, let* $(m_1, \ldots, m_n) \in \mathcal{M}^n$ *be any vector of inputs, and let* $t(n)$ *be the time required to compute* $f(m_1, \ldots, m_n)$*. If* $\mathsf{vk}_\mathcal{P} \leftarrow \mathsf{VerPrep}(\mathsf{vk}, \mathcal{P})$*, then the time required for* $\mathsf{EffVer}(\mathsf{vk}_\mathcal{P}, \Delta, m, \tau)$ *is* $t' = o(t(n))$.

4 A Generic Transformation from Semi-Adaptive to Strong Adaptive Security

In this Section we show a technique that allows one to turn a semi-adaptive unforgeable homomorphic signature into one that satisfies strong adaptive security. Specifically, our main result is stated in the following theorem:

Theorem 1. *If Σ is a semi-adaptive unforgeable fully (resp. leveled) homomorphic signature scheme for boolean circuits, then there exists a strong adaptive unforgeable homomorphic signature scheme $\widehat{\Sigma}$ that supports the same class of functions. Furthermore, if Σ satisfies context-hiding (resp. efficient verification) so does $\widehat{\Sigma}$.*

The core of our result is a general transformation which shows how to combine a semi-adaptive secure scheme Σ together with a semi-adaptive secure scheme Σ_{OR} that supports only OR operations over $\mathbb{Z}_2$. This combination yields a homomorphic signature scheme that is strong adaptive secure and supports the same class of functions supported by Σ.

Clearly, if Σ supports the evaluation of boolean circuits, then Σ_{OR} can be instantiated using Σ. In this case, our result provides a way to bootstrap the security of Σ from semi-adaptive to strong adaptive. This yields our main result above.

In the case where Σ_{OR} cannot be instantiated using Σ (e.g., Σ is not expressive enough), our transformation still provides a recipe to obtain strong adaptive security using a separate OR-homomorphic scheme. However, motivated by the lack of many candidates of OR-homomorphic signature schemes (concretely, [23] is the only available one), we investigated how to obtain a similar transformation by using schemes that have been studied more widely. Our second result is a transformation which can combine a semi-adaptive secure scheme Σ together with a semi-adaptive secure *linearly-homomophic* signature Σ_{LH} that works for messages over a large ring, say $\mathbb{Z}_p$. This combination yields a homomorphic signature scheme that is strong adaptive secure and supports the same class of functions supported by Σ. A limitation of this second transformation is that it applies only to schemes that are leveled homomorphic (i.e., for circuits of bounded depth). On the other hand, the advantage is that strong adaptive security can be obtained by using linearly-homomorphic schemes, a class of constructions that has received significant attention, of which we know many constructions [2,12,3,13,18,4,10,11], most of which are way more efficient in practice than [23]. As for the efficiency of the scheme resulting from our transformations, it basically depends on the efficiency of the scheme one starts from. In the worst case, however, the efficiency loss is comparable to executing the original algorithms twice.

4.1 Strong Adaptive Security from OR-Homomorphic Signatures

Here we present our first transformation. The tools we start from are a homomorphic signature scheme $\Sigma := (\Sigma.\mathsf{KeyGen}, \Sigma.\mathsf{Sign}, \Sigma.\mathsf{Ver}, \Sigma.\mathsf{Eval})$ for a class $\mathcal{C}$ of circuits[8], and a homomorphic signature $\Sigma_{\mathsf{OR}} := (\Sigma_{\mathsf{OR}}.\mathsf{KeyGen}, \Sigma_{\mathsf{OR}}.\mathsf{Sign}, \Sigma_{\mathsf{OR}}.\mathsf{Ver}, \Sigma_{\mathsf{OR}}.\mathsf{Eval})$ that works over message space $\mathbb{Z}_2$ and supports homomorphic OR operations. More precisely, Σ_{OR} must support circuits that are composed only of OR gates and have the same depth as those in $\mathcal{C}$. Using Σ and Σ_{OR} in a black box way, we build a scheme $\widehat{\Sigma}$ which supports evaluation of circuits in $\mathcal{C}$. Moreover, assuming only semi-adaptive security of both Σ and Σ_{OR}, we show that $\widehat{\Sigma}$ is strong adaptive secure.

[8] For the sake of this transformation, they can be either boolean or arithmetic circuits.

$\widehat{\Sigma}.\mathsf{KeyGen}(1^\lambda, \mathcal{L})$**.** Run the key generation algorithms $(\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda, \mathcal{L})$ and $(\mathsf{vk}_{\mathsf{OR}}, \mathsf{sk}_{\mathsf{OR}}) \leftarrow \Sigma_{\mathsf{OR}}.\mathsf{KeyGen}(1^\lambda, \mathcal{L})$, and output $(\hat{\mathsf{vk}}, \hat{\mathsf{sk}}) := ((\mathsf{vk}, \mathsf{vk}_{\mathsf{OR}}), (\mathsf{sk}, \mathsf{sk}_{\mathsf{OR}}))$.

$\widehat{\Sigma}.\mathsf{Sign}(\hat{\mathsf{sk}}, \Delta, \tau, m)$**.** The signing algorithm uses the secret key to compute $\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}, \Delta, \tau, m)$ and $\sigma_{\mathsf{OR}} \leftarrow \Sigma_{\mathsf{OR}}.\mathsf{Sign}(\mathsf{sk}_{\mathsf{OR}}, \Delta, \tau, 0)$, and outputs $\hat{\sigma} := (\sigma, \sigma_{\mathsf{OR}})$.
Note that the OR-homomorphic component σ_{OR} of the signature signs the bit 0. Although the usefulness of this component will become more clear in the security proof, the intuition is that this component keeps track of those labels that are used throughout the computation.

$\widehat{\Sigma}.\mathsf{Eval}(\hat{\mathsf{vk}}, f, \hat{\sigma}_1, \ldots, \hat{\sigma}_n)$**.** We describe the homomorphic evaluation of f in a gate-by-gate fashion, distinguishing the cases of unary and binary gates. One can easily see that the construction generalizes to n-ary gates. Describing the evaluation gate-by-gate is also useful to see that our transformation allows for arbitrary composition of signatures (i.e., running $\widehat{\Sigma}.\mathsf{Eval}$ on outputs of $\widehat{\Sigma}.\mathsf{Eval}$).
At every gate g, one proceeds as follows.

Unary Gates. Let g be an unary gate and let $\hat{\sigma}_1 := (\sigma_1, \sigma_{\mathsf{OR},1})$ be the input. We compute the output signature $\hat{\sigma}_{\mathsf{out}} := (\sigma_{\mathsf{out}}, \sigma_{\mathsf{OR},\mathsf{out}})$ by computing $\sigma_{\mathsf{out}} \leftarrow \Sigma.\mathsf{Eval}(\mathsf{vk}, g, \sigma_1)$ and $\sigma_{\mathsf{OR},\mathsf{out}} \leftarrow \sigma_{\mathsf{OR},1}$. Basically, we evaluate g over the Σ component, while for the OR-homomorphic component we simply evaluate an identity function.

Binary Gates. Let g be a binary gate and let $\hat{\sigma}_1 := (\sigma_1, \sigma_{\mathsf{OR},1})$ and $\hat{\sigma}_2 := (\sigma_2, \sigma_{\mathsf{OR},2})$ be its two inputs. We compute the output signature $\hat{\sigma}_{\mathsf{out}} := (\sigma_{\mathsf{out}}, \sigma_{\mathsf{OR},\mathsf{out}})$ by first evaluating $\sigma_{\mathsf{out}} \leftarrow \Sigma.\mathsf{Eval}(\mathsf{vk}, g, \sigma_1, \sigma_2)$ and then evaluating $\sigma_{\mathsf{OR},\mathsf{out}} \leftarrow \Sigma_{\mathsf{OR}}.\mathsf{Eval}(\mathsf{vk}_{\mathsf{OR}}, \mathsf{OR}, \sigma_{\mathsf{OR},1}, \sigma_{\mathsf{OR},2})$. Basically, we evaluate the binary g over the Σ components, while for the OR-homomorphic components we perform their homomorphic OR.

By proceeding over f in a gate-by-gate fashion, eventually we obtain a signature $\hat{\sigma} := (\sigma, \sigma_{\mathsf{OR}})$, and $\widehat{\Sigma}.\mathsf{Eval}$ returns $\hat{\sigma}$.
At this point, it is worth mentioning that the evaluation algorithm of our transformation generates $(\sigma, \sigma_{\mathsf{OR}})$ such that $\sigma = \Sigma.\mathsf{Eval}(\mathsf{vk}, f, \sigma_1, \ldots, \sigma_n)$ and $\sigma_{\mathsf{OR}} = \Sigma_{\mathsf{OR}}.\mathsf{Eval}(\mathsf{vk}_{\mathsf{OR}}, f_{\mathsf{OR}}, \sigma_{\mathsf{OR},1}, \ldots \sigma_{\mathsf{OR},n})$, where f_{OR} is an "OR version" of the circuit f obtained by changing any unary gate with an identity gate and any binary gate with an OR gate.

$\widehat{\Sigma}.\mathsf{Ver}(\mathsf{vk}, \mathcal{P}, \Delta, m, \hat{\sigma})$**.** Parse $\mathcal{P} = (f, \tau_1, \ldots, \tau_n)$ and $\hat{\sigma} := (\sigma, \sigma_{\mathsf{OR}})$. Next, define $\mathcal{P}_{\mathsf{OR}} := (f_{\mathsf{OR}}, \tau_1, \ldots, \tau_n)$, where f_{OR} is the circuit composed only of OR (and identity) gates, obtained from f as described above. Then check if $\Sigma.\mathsf{Ver}(\mathsf{vk}, \mathcal{P}, \Delta, m, \sigma) = 1$ and $\Sigma_{\mathsf{OR}}.\mathsf{Ver}(\mathsf{vk}_{\mathsf{OR}}, \mathcal{P}_{\mathsf{OR}}, \Delta, 0, \sigma_{\mathsf{OR}}) = 1$. If both the verification runs output 1, then output 1, otherwise output 0.

In the following theorem we show that our generic scheme $\widehat{\Sigma}$ satisfies strong adaptive security, as long as the schemes Σ and Σ_{OR} are only semi-adaptive secure.

Theorem 2. *Assume that Σ is a semi-adaptive secure homomorphic signature scheme for a class of circuits $\mathcal{C}$, and that Σ_{OR} is a semi-adaptive secure homomorphic signature with message space $\mathbb{Z}_2$ and supporting OR circuits. Then the scheme $\widehat{\Sigma}$ described above is a strong-adaptive secure homomorphic signature for $\mathcal{C}$. Furthermore, if both Σ and Σ_{OR} satisfy context-hiding (resp. efficient verification), then so does $\widehat{\Sigma}$.*

Proof. First of all, it is easy to see that the correctness and succinctness of $\widehat{\Sigma}$ are granted by the respective properties of Σ and Σ_{OR}. Second, before showing the security, we provide an intuition about context-hiding and efficient verification (details appear in Appendix A). For context-hiding the idea is that, since every signature of $\widehat{\Sigma}$ consists of a signature of the scheme Σ and a signature of the scheme Σ_{OR}, context-hiding on the two components easily implies context-hiding of the resulting signatures of $\widehat{\Sigma}$.[9] For efficient verification, we observe that the verification algorithm of $\widehat{\Sigma}$ simply consists of running verification of both Σ and Σ_{OR}. Hence, if the latter two algorithms admit an efficient verification mechanism, this mechanism can also be used to obtain efficient verification in $\widehat{\Sigma}$.

Now we focus on the main result, which is the strong adaptive security of $\widehat{\Sigma}$.

Let $\mathbf{Exp}^{\mathsf{strong\text{-}Ad\text{-}UF}}_{\mathcal{A},\widehat{\Sigma}}(\lambda)$ be the strong adaptive security experiment. In this experiment we let T_3 be the event that the adversary $\mathcal{A}$ comes up with a Type 3 Strong forgery. Then we have

$$\mathbf{Adv}^{\mathsf{strong\text{-}Ad\text{-}UF}}_{\mathcal{A},\widehat{\Sigma}}(\lambda) \leq \Pr[\mathbf{Exp}^{\mathsf{strong\text{-}Ad\text{-}UF}}_{\mathcal{A},\widehat{\Sigma}}(\lambda) = 1 \mid \mathsf{T}_3] + \\ + \Pr[\mathbf{Exp}^{\mathsf{strong\text{-}Ad\text{-}UF}}_{\mathcal{A},\widehat{\Sigma}}(\lambda) = 1 \mid \neg\mathsf{T}_3].$$

To show that the above advantage is negligible for any PPT $\mathcal{A}$, in the following lemmas we provide negligible bounds for both the quantities on the right hand side.

Lemma 1. *For any PPT $\mathcal{A}$ there is a PPT $\mathcal{B}$ such that* $\Pr[\mathbf{Exp}^{\mathsf{strong\text{-}Ad\text{-}UF}}_{\mathcal{A},\widehat{\Sigma}}(\lambda) = 1 \mid \neg\mathsf{T}_3] = \mathbf{Adv}^{\mathsf{semi\text{-}Ad\text{-}UF}}_{\mathcal{B},\Sigma}(\lambda)$.

Proof. If T_3 does not happen, in $\mathbf{Exp}^{\mathsf{strong\text{-}Ad\text{-}UF}}_{\mathcal{A},\widehat{\Sigma}}(\lambda)$ the adversary $\mathcal{A}$ can win by coming up with a forgery of either Type 1 or Type 2. In this case, we can easily build a reduction $\mathcal{B}$ which uses a forgery from $\mathcal{A}$ in order to break the semi-adaptive security of Σ.

$\mathcal{B}$ receives a public key vk for Σ from its challenger, generates on its own a key-pair $(\mathsf{vk}_{\mathsf{OR}}, \mathsf{sk}_{\mathsf{OR}})$ for Σ_{OR}, and finally sends $(\mathsf{vk}, \mathsf{vk}_{\mathsf{OR}})$ to $\mathcal{A}$. Whenever $\mathcal{A}$ makes a signing query, $\mathcal{B}$ uses $\mathsf{sk}_{\mathsf{OR}}$ to compute the signature component σ_{OR}, while it obtains σ by forwarding the same query to its challenger. So, it sends to $\mathcal{A}$ the signature $\hat{\sigma} := (\sigma, \sigma_{\mathsf{OR}})$. When $\mathcal{A}$ outputs a Type 1 (resp. Type 2) forgery $(\mathcal{P}^* := (f^*, \tau_1^*, \ldots, \tau_n^*), \Delta^*, m^*, \hat{\sigma}^* := (\sigma^*, \sigma^*_{\mathsf{OR}}))$, $\mathcal{B}$ outputs $(\mathcal{P}^* := (f^*, \tau_1^*, \ldots, \tau_n^*), \Delta^*, m^*, \sigma^*)$ as a Type 1 (resp. Type 2) forgery for the scheme Σ.

Lemma 2. *For any PPT $\mathcal{A}$ there is a PPT $\mathcal{B}'$ such that* $\Pr[\mathbf{Exp}^{\mathsf{strong\text{-}Ad\text{-}UF}}_{\mathcal{A},\widehat{\Sigma}}(\lambda) = 1 \mid \mathsf{T}_3] = \mathbf{Adv}^{\mathsf{semi\text{-}Ad\text{-}UF}}_{\mathcal{B}',\Sigma_{\mathsf{OR}}}(\lambda)$.

Proof. This is the case where we use OR-homomorphic signatures in order to handle Type 3 forgeries. The reduction $\mathcal{B}'$ works as follows.

Key Generation $\mathcal{B}'$ receives a public key $\mathsf{vk}_{\mathsf{OR}}$ of Σ_{OR} from its challenger $\mathcal{C}_{\Sigma_{\mathsf{OR}}}$; then $\mathcal{B}$ runs $(\mathsf{vk}, \mathsf{sk}) \leftarrow \Sigma.\mathsf{KeyGen}(1^\lambda, \mathcal{L})$ and sends $\hat{\mathsf{vk}} = (\mathsf{vk}, \mathsf{vk}_{\mathsf{OR}})$ to $\mathcal{A}$.

Signing Queries Whenever $\mathcal{A}$ asks for a signature on (Δ, τ, m), $\mathcal{B}'$ computes on its own the signature $\sigma \leftarrow \Sigma.\mathsf{Sign}(\mathsf{sk}, \Delta, \tau, m)$ and makes a signing query $(\Delta, \tau, 0)$ to $\mathcal{C}_{\Sigma_{\mathsf{OR}}}$ getting back a signature σ_{OR}. It then sends to $\mathcal{A}$ the signature $\hat{\sigma} := (\sigma, \sigma_{\mathsf{OR}})$.

Forgery Assume that $\mathcal{A}$ outputs a Type 3 Strong forgery $(\mathcal{P}^* = (f^*, \tau_1^*, \ldots, \tau_n^*), \Delta^*, m^*, \hat{\sigma}^*)$. By definition of Type 3 Strong forgery, there exists a *non-empty* subset $\mathcal{J} \subset [n]$ of indices such that, for all $j \in \mathcal{J}$, the label τ_j^* has not been queried by $\mathcal{A}$ during the experiment $\mathbf{Exp}^{\mathsf{strong\text{-}Ad\text{-}UF}}_{\mathcal{A},\widehat{\Sigma}}(\lambda)$.

So, $\mathcal{B}'$ chooses any index $\hat{j} \in \mathcal{J}$ and then makes the following additional signing queries to its challenger: $\{(\Delta^*, \tau_j^*, 0)\}_{j \in \mathcal{J} \setminus \hat{j}}$, $(\Delta^*, \tau^*_{\hat{j}}, 1)$.

[9] It is interesting to note that, even if signatures σ_{OR} only sign 0 (so, they do not contain information on the actual messages), context hiding on Σ_{OR} is still required for the proof to go through.

Let $\hat{\sigma}^* := (\sigma^*, \sigma^*_{\mathsf{OR}})$. By definition of Type 3 Strong forgery we have that $\Sigma_{\mathsf{OR}}.\mathsf{Ver}(\mathsf{vk}_{\mathsf{OR}}, \mathcal{P}^*_{\mathsf{OR}}, \Delta^*, 0, \sigma^*_{\mathsf{OR}}) = 1$, where $\mathcal{P}^*_{\mathsf{OR}} := (f^*_{\mathsf{OR}}, \tau^*_1, \ldots, \tau^*_n)$ and f^*_{OR} is the OR version of the circuit f^*, as computed in the verification algorithm $\widehat{\Sigma}.\mathsf{Ver}$.

$\mathcal{B}'$ returns $(\mathcal{P}^*_{\mathsf{OR}}, \Delta^*, 0, \sigma^*_{\mathsf{OR}})$ as its forgery.

To conclude the proof we claim that the above tuple is a Type 2 forgery against the scheme Σ_{OR}. To see this, first notice that $\mathcal{B}'$ has made signing queries for all the labels $\tau^*_1, \ldots, \tau^*_n$. Second, we claim that $0 \neq y$ where y is the correct output obtained by computing f^*_{OR} on the inputs queried by $\mathcal{B}'$ to its challenger, i.e., obtained by feeding 0 in all input wires of f^*_{OR} labeled by τ^*_i for $i \neq \hat{j}$, and 1 in the input wire labeled with $\tau^*_{\hat{j}}$.

To argue that $y \neq 0$, note that $f_{\mathsf{OR}}(x_1, \ldots, x_n)$ can be written as $\bigvee_{j=1}^n x_j$ and thus $y = x_{\hat{j}} = 1$. Therefore, we have shown that whenever $\mathcal{A}$ breaks the strong adaptive security of $\widehat{\Sigma}$ by producing a Type 3 Strong forgery, $\mathcal{B}'$ can break the semi-adaptive security of Σ_{OR} by producing a Type 2 forgery. This concludes the proof. □

4.2 Strong Adaptive Security from Linearly-Homomorphic Signatures

Here we present our second transformation. This transformation is similar to the one of Section 4.1: it incorporates signatures from a second homomorphic signature scheme in order to handle Type 3 forgeries. However, instead of a OR-homomorphic scheme, here we use a linearly-homomorphic one. More in detail, our constructions takes in a homomorphic signature scheme $\Sigma := (\Sigma.\mathsf{KeyGen}, \Sigma.\mathsf{Sign}, \Sigma.\mathsf{Ver}, \Sigma.\mathsf{Eval})$ that supports circuits of polynomial depth at most d and fan-in 2,[10] and an additive-homomorphic signature $\Sigma_{\mathsf{LH}} := (\Sigma_{\mathsf{LH}}.\mathsf{KeyGen}, \Sigma_{\mathsf{LH}}.\mathsf{Sign}, \Sigma_{\mathsf{LH}}.\mathsf{Ver}, \Sigma_{\mathsf{LH}}.\mathsf{Eval})$ that works over message space $\mathbb{Z}_p$, where $p > 2^d$. Using Σ and Σ_{LH} in a black box way, we build a scheme Σ' which supports the same circuits as Σ, and assuming only semi-adaptive security of Σ and Σ_{LH}, we show that Σ' is strong adaptive secure.

The scheme Σ' is defined as follows:

$\Sigma'.\mathsf{KeyGen}(1^\lambda, \mathcal{L})$. Run both $(\mathsf{vk}, \mathsf{sk}) \leftarrow \Sigma.\mathsf{KeyGen}(1^\lambda, \mathcal{L})$ and $(\mathsf{vk}_{\mathsf{LH}}, \mathsf{sk}_{\mathsf{LH}}) \leftarrow \Sigma_{\mathsf{LH}}.\mathsf{KeyGen}(1^\lambda, \mathcal{L})$, and output $(\mathsf{vk}', \mathsf{sk}') := ((\mathsf{vk}, \mathsf{vk}_{\mathsf{LH}}), (\mathsf{sk}, \mathsf{sk}_{\mathsf{LH}}))$.

$\Sigma'.\mathsf{Sign}(\mathsf{sk}', \Delta, \tau, m)$. The signing algorithm uses sk' to compute $\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}, \Delta, \tau, m)$ and $\sigma_{\mathsf{LH}} \leftarrow \Sigma_{\mathsf{LH}}.\mathsf{Sign}(\mathsf{sk}_{\mathsf{LH}}, \Delta, \tau, 0)$, and outputs $\sigma' := (\sigma, \sigma_{\mathsf{LH}})$.

$\Sigma'.\mathsf{Eval}(\mathsf{vk}', f, \sigma'_1, \ldots, \sigma'_n)$. As in the previous section, we describe the homomorphic evaluation of f in a gate-by-gate fashion, distinguishing the cases of unary and binary gates. At every gate g, one proceeds as follows.

Unary Gates. Let g be an unary gate and let $\sigma'_1 := (\sigma_1, \sigma_{\mathsf{LH},1})$ be the input. We compute the output signature $\sigma'_{\mathsf{out}} := (\sigma_{\mathsf{out}}, \sigma_{\mathsf{LH,out}})$ by computing $\sigma_{\mathsf{out}} \leftarrow \Sigma.\mathsf{Eval}(\mathsf{vk}, g, \sigma_1)$ and $\sigma_{\mathsf{LH,out}} \leftarrow \sigma_{\mathsf{LH},1}$. Basically, we evaluate g over the Σ component, while for the linearly-homomorphic component we simply evaluate an identity function.

Binary Gates. Let g be a binary gate and let $\sigma'_1 := (\sigma_1, \sigma_{\mathsf{LH},1})$ and $\sigma'_2 := (\sigma_2, \sigma_{\mathsf{LH},2})$ be its two inputs. We compute the output signature $\sigma'_{\mathsf{out}} := (\sigma_{\mathsf{out}}, \sigma_{\mathsf{LH,out}})$ by first evaluating $\sigma_{\mathsf{out}} \leftarrow \Sigma.\mathsf{Eval}(\mathsf{vk}, g, \sigma_1, \sigma_2)$ and then evaluating $\sigma_{\mathsf{LH,out}} \leftarrow \Sigma_{\mathsf{LH}}.\mathsf{Eval}(\mathsf{vk}_{\mathsf{LH}}, +, \sigma_{\mathsf{LH},1}, \sigma_{\mathsf{LH},2})$. Basically, we evaluate the binary g over the Σ components, while for the linearly-homomorphic components we perform their homomorphic addition.

[10] We describe the transformation for fan-in 2 only for ease of exposition. It is easy to see that the same technique would work for constant fan-in c setting up $p > c^d$.

By proceeding over f in a gate-by-gate fashion, eventually we obtain a signature $\sigma' := (\sigma, \sigma_{\mathsf{LH}})$, and $\Sigma'.\mathsf{Eval}$ returns σ'.

We note that the evaluation algorithm of our transformation generates $(\sigma, \sigma_{\mathsf{LH}})$ such that $\sigma = \Sigma.\mathsf{Eval}(\mathsf{vk}, f, \sigma_1, \ldots, \sigma_n)$ and $\sigma_{\mathsf{LH}} = \Sigma_{\mathsf{LH}}.\mathsf{Eval}(\mathsf{vk}_{\mathsf{LH}}, f_+, \sigma_{\mathsf{LH},1}, \ldots, \sigma_{\mathsf{LH},n})$, where f_+ is an "additive version" of the circuit f obtained by changing any unary gate with an identity gate and any binary gate with an additive gate.

$\Sigma'.\mathsf{Ver}(\mathsf{vk}, \mathcal{P}, \Delta, m, \sigma')$. Parse $\mathcal{P} = (f, \tau_1, \ldots, \tau_n)$ and $\sigma' := (\sigma, \sigma_{\mathsf{LH}})$. Next, define $\mathcal{P}_+ := (f_+, \tau_1, \ldots, \tau_n)$, where f_+ is the additive circuit obtained from f as described above. Then check if $\Sigma.\mathsf{Ver}(\mathsf{vk}, \mathcal{P}, \Delta, m, \sigma) = 1$ and $\Sigma_{\mathsf{LH}}.\mathsf{Ver}(\mathsf{vk}_{\mathsf{LH}}, \mathcal{P}_+, \Delta, 0, \sigma_{\mathsf{LH}}) = 1$. If both the verification runs output 1, then output 1, otherwise output 0.

In the following theorem we show that our generic scheme Σ' satisfies strong adaptive security, as long as the schemes Σ and Σ_{LH} are only semi-adaptive secure.

Theorem 3. *Assume that Σ is a semi-adaptive secure homomorphic signature scheme for circuits of polynomial depth at least d and fan-in 2, and that Σ_{LH} is a semi-adaptive secure linearly-homomorphic signature scheme whose message space is $\mathbb{Z}_p$, with $p > 2^d$. Then the scheme Σ' described above is a strong-adaptive secure homomorphic signature. Furthermore, if both Σ and Σ_{LH} satisfy context-hiding (resp. efficient verification), then so does Σ'.*

Proof. First of all, it is easy to see that the correctness and succinctness of Σ' are granted by the respective properties of Σ and Σ_{LH}. The inheritance of context-hiding and efficient verification follows in the same way as for the scheme $\widehat{\Sigma}$ of the previous section.

Below we focus on the main result, which is the strong adaptive security of Σ'. The structure of the proof is the same as the proof of Theorem 2; the main difference is the handling of Type 3 Strong forgeries that here is done by using the semi-adaptive security of Σ_{LH}.

Let $\mathbf{Exp}^{\mathsf{strong\text{-}Ad\text{-}UF}}_{\mathcal{A},\Sigma'}(\lambda)$ be the strong adaptive security experiment. In this experiment we let T_3 be the event that the adversary $\mathcal{A}$ comes up with a Type 3 Strong forgery. Then we have

$$\begin{aligned}\mathbf{Adv}^{\mathsf{strong\text{-}Ad\text{-}UF}}_{\mathcal{A},\Sigma'}(\lambda) \leq & \Pr[\mathbf{Exp}^{\mathsf{strong\text{-}Ad\text{-}UF}}_{\mathcal{A},\Sigma'}(\lambda) = 1 \mid \mathsf{T}_3] + \\ & \Pr[\mathbf{Exp}^{\mathsf{strong\text{-}Ad\text{-}UF}}_{\mathcal{A},\Sigma'}(\lambda) = 1 \mid \neg\mathsf{T}_3].\end{aligned}$$

To show that the above advantage is negligible for any PPT $\mathcal{A}$, in the following lemmas we provide negligible bounds for both the quantities on the right hand side.

Lemma 3. *For any PPT $\mathcal{A}$ there is a PPT $\mathcal{B}$ such that* $\Pr[\mathbf{Exp}^{\mathsf{strong\text{-}Ad\text{-}UF}}_{\mathcal{A},\Sigma'}(\lambda) = 1 \mid \neg\mathsf{T}_3] = \mathbf{Adv}^{\mathsf{semi\text{-}Ad\text{-}UF}}_{\mathcal{B},\Sigma}(\lambda)$.

Proof. The proof is essentially the same as that of Lemma 1 and thus is omitted.

Lemma 4. *For any PPT $\mathcal{A}$ there is a PPT $\mathcal{B}'$ such that* $\Pr[\mathbf{Exp}^{\mathsf{strong\text{-}Ad\text{-}UF}}_{\mathcal{A},\Sigma'}(\lambda) = 1 \mid \mathsf{T}_3] = \mathbf{Adv}^{\mathsf{semi\text{-}Ad\text{-}UF}}_{\mathcal{B}',\Sigma_{\mathsf{LH}}}(\lambda)$.

Proof. The reduction $\mathcal{B}'$ works as follows.

Key Generation $\mathcal{B}'$ receives a public key $\mathsf{vk}_{\mathsf{LH}}$ of Σ_{LH} from its challenger $\mathcal{C}_{\Sigma_{\mathsf{LH}}}$; then $\mathcal{B}$ runs $(\mathsf{vk}, \mathsf{sk}) \leftarrow \Sigma.\mathsf{KeyGen}(1^\lambda, \mathcal{L})$ and sends $\mathsf{vk}' = (\mathsf{vk}, \mathsf{vk}_{\mathsf{LH}})$ to $\mathcal{A}$.

Signing Queries Whenever $\mathcal{A}$ asks for a signature on (Δ, τ, m), $\mathcal{B}'$ computes on its own the signature $\sigma \leftarrow \Sigma.\mathsf{Sign}(\mathsf{sk}, \Delta, \tau, m)$ and makes a signing query $(\Delta, \tau, 0)$ to $\mathcal{C}_{\Sigma_{\mathsf{LH}}}$ getting back a signature σ_{LH}. It then sends to $\mathcal{A}$ the signature $\sigma' := (\sigma, \sigma_{\mathsf{LH}})$.

Forgery Assume that $\mathcal{A}$ outputs a Type 3 Strong forgery $(\mathcal{P}^* = (f^*, \tau_1^*, \ldots, \tau_n^*), \Delta^*, m^*, (\sigma^*)')$. By definition of Type 3 Strong forgery, there exists a non-empty subset $\mathcal{J} \subset [n]$ of indices such that, for all $j \in \mathcal{J}$, the label τ_j^* has not been queried by $\mathcal{A}$ during the experiment $\mathbf{Exp}_{\mathcal{A},\Sigma'}^{\mathsf{strong\text{-}Ad\text{-}UF}}(\lambda)$. So, $\mathcal{B}'$ chooses any index $\hat{j} \in \mathcal{J}$ and then makes the following additional signing queries to its challenger: $\{(\Delta^*, \tau_j^*, 0)\}_{j \in \mathcal{J} \setminus \hat{j}}$, $(\Delta^*, \tau_{\hat{j}}^*, 1)$.
Let $(\sigma^*)' := (\sigma^*, \sigma_{\mathsf{LH}}^*)$. By definition of Type 3 Strong forgery we have that $\Sigma'.\mathsf{Ver}(\mathsf{vk}_{\mathsf{LH}}, \mathcal{P}_+^*, \Delta^*, 0, \sigma_{\mathsf{LH}}^*) = 1$, where $\mathcal{P}_+^* := (f_+^*, \tau_1^*, \ldots, \tau_n^*)$ and f_+^* is the additive version of the circuit f^*, as computed in the verification algorithm $\Sigma'.\mathsf{Ver}$.
Finally, $\mathcal{B}'$ returns $(\mathcal{P}_+^*, \Delta^*, 0, \sigma_{\mathsf{LH}}^*)$ as its forgery.

To conclude the proof we claim that the above tuple is a Type 2 forgery against the scheme Σ_{LH}. To see this, first notice that $\mathcal{B}'$ has made signing queries for all the labels $\tau_1^*, \ldots, \tau_n^*$. Second, we claim that $0 \neq y$ where y is the correct output obtained by computing f_+^* on the inputs queried by $\mathcal{B}'$ to its challenger, i.e., obtained by feeding 0 in all input wires of f_+^* labeled by τ_i^* for $i \neq \hat{j}$, and 1 in the input wire labeled with $\tau_{\hat{j}}^*$.

To argue that $y \neq 0$, note that $f_+^*(x_1, \ldots, x_n)$ can be written as $\sum_{j=1}^n \gamma_{\tau_i^*} \cdot x_i$, and (by our choice of the x_i's) $y = \gamma_{\tau_{\hat{j}}^*}$. Moreover, for every i we have that $1 \leq \gamma_{\tau_i^*} < 2^d$. The latter fact is granted by our assumption that the circuits of Σ' (and thus f_+^* too) have fan-in 2 and depth at most d. Hence using our choice of $p > 2^d$ we also have that $y = \gamma_{\tau_{\hat{j}}^*} \neq 0 \mod p$. Therefore, we have shown that whenever $\mathcal{A}$ breaks the strong adaptive security of Σ' by producing a Type 3 Strong forgery, $\mathcal{B}'$ can break the semi-adaptive security of Σ_{LH} by producing a Type 2 forgery. This concludes the proof. □

References

1. S. Agrawal and D. Boneh. Homomorphic MACs: MAC-based integrity for network coding. In M. Abdalla, D. Pointcheval, P.-A. Fouque, and D. Vergnaud, editors, *ACNS 09*, volume 5536 of *LNCS*, pages 292–305. Springer, June 2009.
2. N. Attrapadung and B. Libert. Homomorphic network coding signatures in the standard model. In D. Catalano, N. Fazio, R. Gennaro, and A. Nicolosi, editors, *PKC 2011*, volume 6571 of *LNCS*, pages 17–34. Springer, Mar. 2011.
3. N. Attrapadung, B. Libert, and T. Peters. Computing on authenticated data: New privacy definitions and constructions. In X. Wang and K. Sako, editors, *ASIACRYPT 2012*, volume 7658 of *LNCS*, pages 367–385. Springer, Dec. 2012.
4. N. Attrapadung, B. Libert, and T. Peters. Efficient completely context-hiding quotable and linearly homomorphic signatures. In K. Kurosawa and G. Hanaoka, editors, *PKC 2013*, volume 7778 of *LNCS*, pages 386–404. Springer, Feb. / Mar. 2013.
5. D. Boneh, D. Freeman, J. Katz, and B. Waters. Signing a linear subspace: Signature schemes for network coding. In S. Jarecki and G. Tsudik, editors, *PKC 2009*, volume 5443 of *LNCS*, pages 68–87. Springer, Mar. 2009.
6. D. Boneh and D. M. Freeman. Homomorphic signatures for polynomial functions. In K. G. Paterson, editor, *EUROCRYPT 2011*, volume 6632 of *LNCS*, pages 149–168. Springer, May 2011.
7. D. Boneh and D. M. Freeman. Linearly homomorphic signatures over binary fields and new tools for lattice-based signatures. In D. Catalano, N. Fazio, R. Gennaro, and A. Nicolosi, editors, *PKC 2011*, volume 6571 of *LNCS*, pages 1–16. Springer, Mar. 2011.
8. D. Catalano and D. Fiore. Practical homomorphic MACs for arithmetic circuits. In T. Johansson and P. Q. Nguyen, editors, *EUROCRYPT 2013*, volume 7881 of *LNCS*, pages 336–352. Springer, May 2013.
9. D. Catalano, D. Fiore, R. Gennaro, and L. Nizzardo. Generalizing homomorphic MACs for arithmetic circuits. In H. Krawczyk, editor, *PKC 2014*, volume 8383 of *LNCS*, pages 538–555. Springer, Mar. 2014.
10. D. Catalano, D. Fiore, R. Gennaro, and K. Vamvourellis. Algebraic (trapdoor) one-way functions and their applications. In A. Sahai, editor, *TCC 2013*, volume 7785 of *LNCS*, pages 680–699. Springer, Mar. 2013.

11. D. Catalano, D. Fiore, and L. Nizzardo. Programmable hash functions go private: Constructions and applications to (homomorphic) signatures with shorter public keys. In *CRYPTO 2015*. Springer, 2015. Full version at: https://eprint.iacr.org/2015/826.
12. D. Catalano, D. Fiore, and B. Warinschi. Adaptive pseudo-free groups and applications. In K. G. Paterson, editor, *EUROCRYPT 2011*, volume 6632 of *LNCS*, pages 207–223. Springer, May 2011.
13. D. Catalano, D. Fiore, and B. Warinschi. Efficient network coding signatures in the standard model. In M. Fischlin, J. Buchmann, and M. Manulis, editors, *PKC 2012*, volume 7293 of *LNCS*, pages 680–696. Springer, May 2012.
14. D. Catalano, D. Fiore, and B. Warinschi. Homomorphic signatures with efficient verification for polynomial functions. In J. A. Garay and R. Gennaro, editors, *CRYPTO 2014, Part I*, volume 8616 of *LNCS*, pages 371–389. Springer, Aug. 2014.
15. M. Chase, M. Kohlweiss, A. Lysyanskaya, and S. Meiklejohn. Malleable signatures: New definitions and delegatable anonymous credentials. In *2014 IEEE 27th Computer Security Foundations Symposium*, pages 199–213. IEEE, 2014.
16. Y. Desmedt. Computer security by redefining what a computer is. NSPW, 1993.
17. K. Elkhiyaoui, M. Önen, and R. Molva. Online-offline homomorphic signatures for polynomial functions. Cryptology ePrint Archive, Report 2015/954, 2015. `http://eprint.iacr.org/`.
18. D. M. Freeman. Improved security for linearly homomorphic signatures: A generic framework. In M. Fischlin, J. Buchmann, and M. Manulis, editors, *PKC 2012*, volume 7293 of *LNCS*, pages 697–714. Springer, May 2012.
19. R. Gennaro, C. Gentry, and B. Parno. Non-interactive verifiable computing: Outsourcing computation to untrusted workers. In T. Rabin, editor, *CRYPTO 2010*, volume 6223 of *LNCS*, pages 465–482. Springer, Aug. 2010.
20. R. Gennaro, J. Katz, H. Krawczyk, and T. Rabin. Secure network coding over the integers. In P. Q. Nguyen and D. Pointcheval, editors, *PKC 2010*, volume 6056 of *LNCS*, pages 142–160. Springer, May 2010.
21. R. Gennaro and D. Wichs. Fully homomorphic message authenticators. In K. Sako and P. Sarkar, editors, *ASIACRYPT 2013, Part II*, volume 8270 of *LNCS*, pages 301–320. Springer, Dec. 2013.
22. S. Goldwasser, S. Micali, and R. L. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM J. Comput.*, 17(2):281–308, 1988.
23. S. Gorbunov, V. Vaikuntanathan, and D. Wichs. Leveled fully homomorphic signatures from standard lattices. In *47th ACM STOC*. ACM Press, 2015.
24. R. Johnson, D. Molnar, D. X. Song, and D. Wagner. Homomorphic signature schemes. In B. Preneel, editor, *CT-RSA 2002*, volume 2271 of *LNCS*, pages 244–262. Springer, Feb. 2002.

A Context Hiding and Efficient Verification of $\widehat{\Sigma}$

Lemma 5. *Assume that* $\Sigma := (\Sigma.\mathsf{KeyGen}, \Sigma.\mathsf{Sign}, \Sigma.\mathsf{Ver}, \Sigma.\mathsf{Eval})$ *and* $\Sigma_{\mathsf{OR}} := (\Sigma_{\mathsf{OR}}.\mathsf{KeyGen}, \Sigma_{\mathsf{OR}}.\mathsf{Sign}, \Sigma_{\mathsf{OR}}.\mathsf{Ver}, \Sigma_{\mathsf{OR}}.\mathsf{Eval})$ *are an homomorphic and OR homomorphic signature scheme respectively, and let both* Σ *and* Σ_{OR} *support context hiding in the sense of Definition 6. Then, the homomorphic signature scheme* $\widehat{\Sigma}$*, output of the black box transformation described in Section 4.1 supports context hiding as well.*

Proof. Since Σ and Σ_{OR} support context hiding, we know that by definition there exist additional PPT procedures $\Sigma.\mathsf{Hide}$, $\Sigma.\mathsf{HVerify}$, $\Sigma_{\mathsf{OR}}.\mathsf{Hide}$, $\Sigma_{\mathsf{OR}}.\mathsf{HVerify}$ which satisfy correctness and unforgeability in the sense of Definition 6. Moreover, there exist two simulators $\Sigma.\mathsf{Sim}$ and $\Sigma_{\mathsf{OR}}.\mathsf{Sim}$ that satisfy the context hiding security property. What we do is thus to show how to build the corresponding procedures $\widehat{\Sigma}.\mathsf{Hide}$, $\widehat{\Sigma}.\mathsf{HVerify}$ and the simulator $\widehat{\Sigma}.\mathsf{Sim}$ for the homomorphic signature scheme $\widehat{\Sigma}$. Since any signature in $\widehat{\Sigma}$ is of the form $\hat{\sigma} := (\sigma, \sigma_{\mathsf{OR}})$, we define $\widehat{\Sigma}.\mathsf{Hide}(\hat{\mathsf{vk}}, m, \hat{\sigma}) := (\Sigma.\mathsf{Hide}(\mathsf{vk}, m, \sigma), \Sigma_{\mathsf{OR}}.\mathsf{Hide}(\mathsf{vk}_{\mathsf{OR}}, 0, \sigma_{\mathsf{OR}}))$ and $\widehat{\Sigma}.\mathsf{HVerify}(\hat{\mathsf{vk}}, \mathcal{P}, \Delta, m, \hat{\sigma}) := \Sigma.\mathsf{HVerify}(\mathsf{vk}, \mathcal{P}, \Delta, m, \sigma) \wedge \Sigma_{\mathsf{OR}}.\mathsf{HVerify}(\mathsf{vk}_{\mathsf{OR}}, \mathcal{P}_{\mathsf{OR}}, \Delta, 0, \sigma_{\mathsf{OR}})$.

It is not hard to see that correctness and unforgeability with respect to $\widehat{\Sigma}.\mathsf{Hide}$ and $\widehat{\Sigma}.\mathsf{HVerify}$ follows from the same properties of the corresponding procedures of Σ and Σ_{OR}. Similarly, we define the simulator $\widehat{\Sigma}.\mathsf{Sim}$ as $\widehat{\Sigma}.\mathsf{Sim}(\hat{\mathsf{sk}}, \mathcal{P}, \Delta, m) := (\Sigma.\mathsf{Sim}(\mathsf{sk}, \mathcal{P}, \Delta, m), \Sigma_{\mathsf{OR}}.\mathsf{Sim}(\mathsf{sk}_{\mathsf{OR}}, \mathcal{P}_{\mathsf{OR}}, \Delta, 0))$. The context hiding of this simulator follows immediately from that of the two simulators that are used to define it.

Remark 1. It is interesting to mention that, although in the construction the signature component σ_{OR} signs 0 (and is thus independent of the actual message), a context hiding property for $\varSigma_{\mathsf{OR}}$ is still needed for the proof to go through. One reason for this is that in a homomorphic signature scheme signatures may change form (and distribution) after applying the evaluation algorithm, and context hiding actually requires that such output distribution is simulatable. That is why we need context hiding for $\varSigma_{\mathsf{OR}}$. Nevertheless, we note that in our specific case where the input messages are fixed, much simpler context hiding mechanisms could be applied, such as requiring signatures to be deterministically generated (e.g., using a PRF) and letting the simulator simply reconstruct such signatures and apply evaluation.

Lemma 6. *Assume that* $\varSigma := (\varSigma.\mathsf{KeyGen}, \varSigma.\mathsf{Sign}, \varSigma.\mathsf{Ver}, \varSigma.\mathsf{Eval})$ *and* $\varSigma_{\mathsf{OR}} := (\varSigma_{\mathsf{OR}}.\mathsf{KeyGen}\varSigma_{\mathsf{OR}}.\mathsf{Sign}, \varSigma_{\mathsf{OR}}.\mathsf{Ver}, \varSigma_{\mathsf{OR}}.\mathsf{Eval})$ *are an homomorphic and OR homomorphic signature scheme respectively, and let both* $\varSigma$ *and* $\varSigma_{\mathsf{OR}}$ *satisfy efficient verification in the sense of Definition 7. Then, the homomorphic signature scheme* $\widehat{\varSigma}$*, output of the black box transformation described in Section 4.1 satisfies efficient verification as well.*

Proof. Since $\varSigma$ and $\varSigma_{\mathsf{OR}}$ satisfy efficient verification, there exist additional algorithms $(\varSigma.\mathsf{VerPrep}, \varSigma.\mathsf{EffVer})$ and $(\varSigma_{\mathsf{OR}}.\mathsf{VerPrep}, \varSigma_{\mathsf{OR}}.\mathsf{EffVer})$ which satisfy correctness and amortized efficiency in the sense of Definition 7. What we do is thus to show how to build the corresponding algorithms $(\widehat{\varSigma}.\mathsf{VerPrep}, \widehat{\varSigma}.\mathsf{EffVer})$ for $\widehat{\varSigma}$. Since any signature in $\widehat{\varSigma}$ is of the form $\hat{\sigma} := (\sigma, \sigma_{\mathsf{OR}})$, we define $\widehat{\varSigma}.\mathsf{VerPrep}(\hat{\mathsf{vk}}, \mathcal{P})$ as the algorithm that runs $\varSigma.\mathsf{VerPrep}(\mathsf{vk}, \mathcal{P})$ and $\varSigma_{\mathsf{OR}}.\mathsf{VerPrep}(\mathsf{vk}_{\mathsf{OR}}, \mathcal{P}_{\mathsf{OR}})$, and outputs $\hat{\mathsf{vk}}_{\mathcal{P}} := (\mathsf{vk}_{\mathcal{P}}, \mathsf{vk}_{\mathsf{OR},\mathcal{P}_{\mathsf{OR}}})$. Then we define $\widehat{\varSigma}.\mathsf{EffVer}(\hat{\mathsf{vk}}_{\mathcal{P}}, \varDelta, m, \hat{\sigma})$ as the algorithm that outputs 1 if and only if both $\varSigma.\mathsf{EffVer}(\mathsf{vk}_{\mathcal{P}}, \varDelta, m, \sigma) = 1$ and $\varSigma_{\mathsf{OR}}.\mathsf{EffVer}(\mathsf{vk}_{\mathsf{OR},\mathcal{P}_{\mathsf{OR}}}, \varDelta, 0, \sigma_{\mathsf{OR}}) = 1$. It is trivial to see that the new algorithms are correct if so are the efficient verification algorithms of $\varSigma$ and $\varSigma_{\mathsf{OR}}$. The efficient verification property instead follows from the fact that $\widehat{\varSigma}.\mathsf{EffVer}$ simply runs two algorithms that are guaranteed to have each running time at most $o(t(n))$.