Bounded Fully Homomorphic Signature Schemes

Xiang Xie[1], Rui Xue[2]

[1] Trusted Computing and Information Assurance Laboratory
Institute of Software, Chinese Academy of Sciences
[2] The State Key Laboratory of Information Security
Institute of Information Engineering, Chinese Academy of Sciences
xiexiang,@tca.iscas.ac.cn, xuerui@iie.ac.cn

Abstract. Homomorphic signatures enable anyone to publicly perform computations on signed data and produce a compact tag to authenticate the results. In this paper, we construct two bounded fully homomorphic signature schemes, as follows.

- For any two polynomials $d = d(\lambda), s = s(\lambda)$, where λ is the security parameter. Our first scheme is able to evaluate any circuit on the signatures, as long as the depth and size of the circuit are bounded by d and s, respectively. The construction relies on indistinguishability obfuscation and injective (or polynomially bounded pre-image size) one-way functions.
- The second scheme, removing the restriction on the size of the circuits, is an extension of the first one, with succinct verification and evaluation keys. More specifically, for an a-prior polynomial $d = d(\lambda)$, the scheme allows to evaluate any circuit on the signatures, as long as the depth of the circuit is bounded by d. This scheme is based on differing-inputs obfuscation and collision-resistant hash functions and relies on a technique called recording hash of circuits.

Both schemes enjoy the composition property. Namely, outputs of previously derived signatures can be re-used as inputs for new computations. The length of derived signatures in both schemes is independent of the size of the data set. Moreover, both constructions satisfy a strong privacy notion, we call *semi-strong context hiding*, which requires that the derived signatures of evaluating any circuit on the signatures of two data sets are *identical* as long as the evaluations of the circuit on these two data sets are the same.

Keywords: homomorphic signature, indistinguishability obfuscation, differing-inputs obfuscation

1 Introduction

Cloud computing allows users with limited storage capacity to securely outsource their data to a remote server. Meanwhile, it also allows the server to reliably perform computations over the data. Recent groundbreaking work of *fully homomorphic encryption* [38] shows how to maintain *privacy* of outsourced data in this setting. In this work, we focus on the orthogonal question of providing *integrity/authenticity* of outsourced data.

More specifically, assume Alice outsources her data $(m_1, ..., m_k)$ along with signatures for the messages to an (untrusted) server. At some later point in time, Bob queries the server for a computation of a circuit on the data, denoted by $C(m_1, ..., m_k)$. The server computes $m \leftarrow C(m_1, ..., m_k)$ and sends it to Bob. Now the problem is that Bob wants to be sure that m is indeed the value obtained by evaluating C on Alice's data. A trivial solution would be the server sends all Alice's data along with the signatures to Bob, Bob first verifies each message by checking the signature and then computes C on them. However, this solution vanishes the advantage of outsourcing and requires large bandwidth of Bob to download the whole data. Further, if Alice requests the data be kept private, with only m to be public. This trivial solution does not even work.

Homomorphic signatures [44,18] can be used to address this problem. With a homomorphic signature scheme, Alice signs messages $m_1, ..., m_k$, producing signatures $\sigma_1, ..., \sigma_k$ and uploads the message-signature pairs to the server. When Bob requires some computation C on Alice's messages, the server (publicly) produces a new signature for the value $m \leftarrow C(m_1, ..., m_k)$. Validation of the

signature asserts that m is indeed the result by applying C on Alice's data. An important feature of homomorphic signatures is that the length of the produced signature is much shorter than k (say sublinear in k). And also, the derived signature should not reveal anything about the data set $m_1, ..., m_k$ beyond what can be learned from m.

The previous homomorphic signature schemes only support restricted class of functions such as linear functions (*e.g.* [17,36,19,9]) or constant degree polynomials [18]. Recently, Gennaro and Wichs [37] formally defined and constructed the secret-key analogue of homomorphic signature, that is *homomorphic message authenticators*. Their construction makes use of fully homomorphic encryptions to evaluate arbitrary circuits. Subsequent works in [24,25] also gave practical homomorphic message authentication schemes for bounded circuits. However, constructing fully homomorphic signatures or even homomorphic signatures for more complex functions remains an important open problem.

1.1 Our Contributions

The main contributions of this work are two bounded fully homomorphic signature schemes. More specifically, let $\mathcal{M}$ be the message space with *polynomial size*. For any two polynomials $d = d(\lambda), s = s(\lambda)$ in security parameter λ, our first scheme is able to evaluate any such circuit over $\mathcal{M}$ on the signatures that the depth and size of the circuit are bounded by d and s, respectively. This scheme also supports composition in the sense that the outputs of authenticated computations can also be used as inputs for new computations. Our first scheme relies on *indistinguishability obfuscation* and injective (or polynomially bounded pre-image size) one-way functions.

Our second scheme is extended from the first one, and removes the restriction on the size of the circuits by applying *differing-inputs obfuscation* and collision-resistant hash functions. For any priori given polynomial $d = d(\lambda)$, the scheme enables to evaluate any circuit on the signatures as long as the depth of the circuit is bounded by d. This scheme has succinct evaluation and verification keys, and still supports composition.

Both schemes are selectively unforgeable, *i.e.* the adversary should claim the message and circuit he wants to forge ahead of time. Although our unforgeability model is selective, it is strengthened in following two aspects. (I) In the signature query phase, almost all the previous models (except the one in [32]) require the adversary to query the signatures of all the messages belonging to one data set at the same time, while our model allows the adversary to *adaptively* query one message at a time, and even intersperse queries from different data sets. (II) The adversary in our model is allowed to specify the tag (used to bind together the messages from the same data set) on his own choice. While all previous models (for publicly verifiable schemes) require the tag be unpredictable and chosen by the challenger uniformly at random.

The length of derived signatures in our schemes is λ, which is independent of the size of the data sets. Moreover, our signature schemes are deterministic (the signing algorithms are deterministic) and satisfy a privacy notion we call *semi-strong context hiding*, which is weaker than *strong context hiding* [2] and stronger than *weak context hiding* [19,18]. Informally speaking, strong context hiding requires derived signatures be distributed as independent fresh signatures on the same message. Therefore, it even hides the operations on the signatures. Weak context hiding only requires derived signatures do not reveal information of the original messages beyond what is learned from the derived messages and *assumes the original signatures are not public*. Semi-strong context hiding does not hide the fact that derivation took place, however it hides the information of the original messages even the original signatures are exposed (which commonly happens in real environments). More specifically, our schemes satisfy the following property: for any circuit C (in the admissible circuit family) and any two tuples of messages $(m_1, ..., m_k)$, $(m'_1, ..., m'_k)$, the derived signatures of evaluating C on the signatures of these two tuples of messages are *exactly the same* as long as $C(m_1, ..., m_k) = C(m'_1, ..., m'_k)$.

1.2 Overview of Our Techniques

The basic idea of this work is inspired from the signature scheme proposed by Sahai and Waters [51]. In that construction, the signatures are essentially the classical PRF MACs. In order to make it publicly verifiable, they created an obfuscated verification program that checks the MACs. Sahai and Waters [51] showed that this construction is selectively unforgeable under indistinguishability obfuscation and one-way functions. Informally speaking, indistinguishability obfuscation requires that obfuscations of any two distinct (equal-size) programs that implement *identical* functionalities be computationally indistinguishable.

Our first idea comes from that one could create an obfuscated re-signing program for each gate in the circuits and put them in the evaluation key. More specifically, the re-signing programs work as follows. Taking an addition gate for example. With input two message-signature-circuit tuples $(m_1, \sigma_1, C_1), (m_2, \sigma_2, C_2)$, the program first checks the validity of each signature (*i.e.* checks the PRF). It then outputs the computation of the PRF on $m_1 + m_2$ along with the description of $C_1 + C_2$ as a derived signature. Here, (m_1, m_2) and (σ_1, σ_2) are derived messages and signatures by evaluating (C_1, C_2) on the original messages and signatures, respectively.

At the first sight, this construction works well since the derived signature binds the derived message and the circuit together. However, the major technique challenge is to formally prove that the scheme is unforgeable. We extend the "punctured program" method of Sahai and Waters [51] and make use of puncturable PRFs [51]. A punctured PRF key can compute all the inputs except ones in a particular polynomial-size punctured set. Let m^*, C^* be the challenge message and circuit. In the security proof, we will excise out the information of the evaluation of the PRF in the punctured set $\{m^* \| C^*\}$, *i.e.* $\mathsf{F.Eval}(K, m^* \| C^*)$, in the programs. In the verification program, the puncturing process is direct by using a punctured key on the set $\{m^* \| C^*\}$ as done in [51]. However, there is no method to remove $\mathsf{F.Eval}(K, m^* \| C^*)$ in the re-signing programs, because itself is the output of the program when the derived message is m^* and the circuit is C^*.

Before moving on, we first informally recall what is a valid forgery in a homomorphic signature scheme. Let m^*, C^* be the challenge message and circuit the adversary wants to forge, a basic requirement is that for any messages $(\tilde{m}_1, ..., \tilde{m}_k)$ the adversary queries to the signing oralce must satisfy that $m^* \neq C^*(\tilde{m}_1, ..., \tilde{m}_k)$. Otherwise, the scheme is trivially unsecure, because the adversary can evaluate C^* on the queried signatures to obtain a valid signature on m^*. For simplicity of exposition, we now consider C^* ends up with an addition gate. For two tuples $(m_1, \sigma_1, C_1), (m_2, \sigma_2, C_2)$, if σ_1, σ_2 both pass the verification and such that $m_1 + m_2 = m^*$ and $C_1 + C_2 = C^*$, then it must be the case that $m_1 \neq C_1(\tilde{m}_1, ..., \tilde{m}_k)$ or $m_2 \neq C_2(\tilde{m}_1, ..., \tilde{m}_k)$ for every query $(\tilde{m}_1, ..., \tilde{m}_k)$. Suppose not, there exists at least one query $(\tilde{m}_1, ..., \tilde{m}_k)$ such that $m^* = m_1 + m_2 = (C_1 + C_2)(\tilde{m}_1, ..., \tilde{m}_k) = C^*(\tilde{m}_1, ..., \tilde{m}_k)$ which violates the rules in the experiment. Therefore, σ_1 or σ_2 must be a valid forgery for (m_1, C_1) or (m_2, C_2) respectively.

The observation above is still not sufficient to make the security proof go through. In additionally, we modify our scheme by evaluating the circuits level by level. For any two polynomials $d = d(\lambda), s = s(\lambda)$, let $\mathcal{C}_{d,s}$ be the set of circuits with depth and size bounded by d and s, respectively. The scheme first generates $d+1$ PRF keys $K_0, K_1, ..., K_d$ and uses K_0 as the signing key. For each level $1 \leq j \leq d$, we create an evaluation program for each gate and publish the indistinguishability obfuscation of these programs as the evaluation key. These evaluation programs first check the validity of the input signatures using K_{j-1}, and then re-sign (compute the PRF) the evaluation of the messages and the description of the new circuit using K_j.

This chain structure allows us to prove the security by induction on the depth d. Denote $\Pi_{j,n}$ the homomorphic scheme for circuits in $\mathcal{C}_{j,s}$ for $0 \leq j \leq d$. Notice that $\Pi_{0,s}$ is essentially the signature scheme presented by Sahai and Waters [51], since we do not define evaluation programs in this case.

Assuming $\Pi_{d-1,s}$ is selectively unforgeable for circuits in $\mathcal{C}_{d-1,s}$, we show that $\Pi_{d,s}$ is selectively unforgeable for circuits in $\mathcal{C}_{d,s}$.

Roughly speaking, for challenge message m^* and circuit $C^* \in \mathcal{C}_{d,s}$, we assume the depth of C^* is d and C^* ends up with an addition gate. We would like to point out that if the depth of C^* is less than d, then there is a direct reduction from $\Pi_{d-1,s}$. One can also use a similar analysis when the output gate of C^* is another gate not addition. In each evaluation program (for different gates) at depth d, we modify the program to puncture out the point $\{m^*\|C^*\}$ of the PRF indexed by key K_d. For an input $((m_1, \sigma_1, C_1), (m_2, \sigma_2, C_2))$, the modified program first checks the validity of the signatures σ_1, σ_2. After the verification (if they both pass), the modified program simply aborts if $m_1 \diamond m_2 = m^*$ and $C_1 \diamond C_2 = C^*$; otherwise it uses a punctured key K'_d on the set $\{m^*\|C^*\}$ to compute the PRF on $(m_1 \diamond m_2 \| C_1 \diamond C_2)$. Where $\diamond$ is the gate corresponding to this evaluation program. In the programs that *do not* evaluate an addition gate, this modification does not change the functionality. This is because K_d and K'_d compute the same PRF value on all inputs except $m^*\|C^*$, and the abort condition in the modified program will never be triggered since $C_1 \diamond C_2 \neq C^*$ when $\diamond$ is not an addition gate. Thus the property of indistinguishability obfuscator guarantees that efficient adversaries will not detect these modifications. In the program that evaluates the addition gate, there indeed exists differing inputs which satisfy that σ_1 and σ_2 pass the verification, $m_1 + m_2 = m^*$, $C_1 + C_2 = C^*$ and C_1, C_2 are circuits in $\mathcal{C}_{d-1,s}$

From the analysis before, we conclude that one of the signatures in a differing input must be a valid forgery for some circuit in $\Pi_{d-1,s}$. This naturally leads us to use differing-inputs obfuscation which says that if an efficient adversary can distinguish the obfuscations of two circuits, then there exists an efficient extractor outputs a differing input. Taking a closer look to the differing inputs, we show that the number of inputs the two programs differ is at most $2|\mathcal{M}|$, where $\mathcal{M}$ is the message space. Fortunately, Boyle, Chung and Pass [21] said that an indistinguishability obfuscator is sufficient to extract differing inputs when this number is polynomial. Therefore, we fix the message size to be polynomial and make our first scheme use indistinguishability obfuscation again.

However, in the proof of the induction, we actually face another problem. Notice that in order to simulate the evaluation programs at depth d, the simulator has to know the PRF key K_{d-1} (for the verification of the input signatures) which is embedded in the evaluation programs in $\Pi_{d-1,s}$. To address this problem, we also create verification programs on each level, and obfuscate these programs. Instead of directly using the PRF key to verify the validity of the input signatures, each evaluation program uses the obfuscated verification program at the previous level. Note that these obfuscated verification programs can be given in $\Pi_{d-1,s}$. The evaluation key in our scheme now actually consists of obfuscated evaluation programs embedded with other obfuscations.

A drawback of the above scheme is that the admissible circuits have restrictions both on size and depth. Further, the size of evaluation and verification keys is linear in both the upper bounds d, s of the circuits depth and size. This is because the evaluation and verification programs are needed in each level to make the proof go through, and they directly take the descriptions of the circuits as inputs. We then use differing-inputs obfuscation and collision-resistant hash functions to remove the restriction on the size which yields the second scheme. This scheme has succinct evaluation and verification keys with size only depending on the upper bound d. In order to get this scheme, we put forward a technique called recording hash of circuits. Intuitively, the recording hash of a circuit C maps it to a string of fixed length while recording the computation of C. Recording hash suits our construction very well, because when evaluating the signatures we need to bind the computation of the circuits while compressing their size. We expect it to have other applications.

1.3 Related Work

Homomorphic Signatures and Message Authenticators. Recently, many papers have considered the problem of homomorphic signatures and message authentications (private verification) for restricted functions (linear functions). This line of research was initiated by Johnson *et.al.* [44] and became very popular because of the important application to *network coding*. Constructions for linear functions have been proposed in [1,17,36,8,19,27,9,28,32,10,26]. Other similar notions called *proofs of data* and *retrievability* for oursourced storage have also been considered in [6,52,7,31].

The first homomorphic signature scheme considered a larger class of homomorphisms beyond linear functions was given by Boneh and Freeman [18], who showed a realization for constant degree polynomial functions in the random oracle model. In that work, they also presented a general definition along the same lines as the definition we use in this work, and posed an interesting open problem to construct fully homomorphic signatures for arbitrary functions.

Very recently, Gennaro and Wichs [37] proposed the first fully homomorphic message authentication scheme which relies on fully homomorphic encryption schemes. Subsequent works in [24] and [25] also showed how to get practical homomorphic message authentication schemes for some bounded circuits. However, these schemes only provide private verification and barely consider the privacy of the data sets.

Succinct Non-Interactive Argument of Knowledge. One possible solution to construct homomorphic signatures is to rely on computationally sound (CS) proofs [47] or, more generally, *Succinct Non-interactive ARguments of Knowledge* (SNARKs) [16]. In a nutshell, SNARK allows to create a short argument for any NP statement. The length of the argument is independent of the statement and witness. However, constructing SNARKs is known to require the random oracle model [47] or *non-falsifiable* assumptions [40]. Further, the derived signatures (the arguments) using SNARK are not as compact as the original signatures, while the length of the derived and original signatures is the same in our schemes. We note that although assuming the existence of indistinguishability/differing-inputs obfuscator is probably a non-falsifiable assumption, there is still a possibility that our schemes rely on falsifiable assumptions if constructions of indistinguishability/differing-inputs obfuscator from falsifiable assumptions are given. A recent progress on this aspect was made in [39].

Verifiable Computation and Memory Delegation. Other works considered similar notions such as *verifiable computation* [42,5,29,35,15,49,48] and *memory delegation* [30]. In the setting of verifiable computation, one wants to delegate computationally heavy tasks to a power server while maintaining the ability to verify the result efficiently. The aim of verifiable computation is trying to minimize the computation, while in our setting is to minimize communication. Memory delegation allows a client to outsource a large amount of data to an untrusted server and later can efficiently verify computations on data. Current constructions require the client to keep a state and have a time-consuming offline phase where the running time of the client is polynomial on the size of the data.

2 Preliminaries

The security parameter throughout the paper is λ. For an integer $n = n(\lambda)$, the integer set $\{1, ..., n\}$ is denoted by $[n]$. A function $\mathrm{negl}(\lambda)$ is *negligible*, if it vanishes faster than the inverse of any polynomial in λ. For two bit string x, y, the concatenation of x and y is denoted by $x\|y$. PPT means probabilistic polynomial time.

2.1 Arithmetic Circuits

We provide a brief overview of arithmetic circuits and refer the reader to [53] for more details. An arithmetic circuit C over a field $\mathbb{F}$ and a set of variables $X = \{x_1, ..., x_k\}$ is a directed acyclic graph as follows. The nodes in the graph are called *gates*. Gates with in-degree 0 are called input gates while gates with out-degree 0 are called output gates. Each input is labeled by either a variable from X or a constant in $\mathbb{F}$. Gates with in-degree and out-degree greater than 0 are called internal gates. Gates labeled with $\times$ are called product gates, while gates labeled with $+$ are called addition gates. In this paper, we focus on circuits with a single output node and where the in-degree of each product and addition gate is 2. The *size* of a circuit is the number of its gates. The *depth* of a circuit is the longest path from input to output.

In this work, we consider arithmetic circuits on fields with *polynomial size*. We restrict without loss of generality that product gates and addition gates do not get inputs labeled by constants. This can be done as follows: One could view a circuit $C(x_1, ..., x_k)$ over $\mathbb{F}$ as a multivariable polynomial $f(x_1, ..., x_k)$ over $\mathbb{F}$. Since the size of $\mathbb{F}$ is polynomial, each monomial in f with coefficient $a \in \mathbb{F}$ can be written as a summation of a such monomials, and the size of this decomposition form will not increase too much. Therefore, one could set $f(x_1, ..., x_k) = f'(x_1, ..., x_k) + c$, where c is the constant term of f and the coefficients of f' are all 1s. Notice that $f'(x_1, ..., x_k)$ could be implemented by a circuit over $\mathbb{F}$ with all the input gates labeled by the variables in X. Moreover, in homomorphic signatures, authenticating whether a message is derived from the data set by evaluating f is equivalent to authenticate whether it is derived from the same data set by evaluating f', because f is public. We would like to point out that, we restrict the circuits only to avoid duplicative work and save space, and we could also take general product and addition gates into account and the constructions and proofs are followed in a very direct way. In an extreme case, one could think of the field as $\mathbb{F}_2$ and the circuits we consider are actually boolean circuits.

We also insist that each wire in the circuit over $\mathbb{F}$ connects gates at consecutive level. If this is not the case, we can then add identity gates to make it so. Therefore, we consider circuits with product gates ($\times$), addition gates ($+$) and identity gates ($=$) as internal gates. We also define the size of a circuit C, denoted by $|C|$, as the length of the binary representation of C. This definition of circuit size is (polynomially) equivalent to the definition before. For two polynomials $d = d(\lambda)$, $s = s(\lambda)$, let $\mathcal{C}_{d,s}$ be the family of circuits over $\mathbb{F}$ with depth and size bounded by d and s, respectively. For any circuit $C \in \mathcal{C}_{d,s}$ with $|C| < s$, we can represent it with a bit string of length s by padding the circuit. For simplicity of exposition, we do not explicitly use this padding.

2.2 Recording Hash of Circuits

In this subsection, we put forward the notion of recording hash of circuits. Let $h : \{0,1\}^* \to \{0,1\}^\lambda$ be a hash function. Let $\mathcal{C}$ be a family of circuits over field $\mathbb{F}$. For any $C \in \mathcal{C}$, we define the recording hash of C with respect to h, denoted by $\mathsf{RH}_{h,\mathcal{C}}(C)$, recursively as follows:

$$\mathsf{RH}_{h,\mathcal{C}}(C) = \begin{cases} h(C), & \text{if the depth of } C \text{ is 0;} \\ h\Big(\mathsf{RH}_{h,\mathcal{C}}(C_1)\| + \|\mathsf{RH}_{h,\mathcal{C}}(C_2)\Big), & \text{if } C = C_1 + C_2; \\ h\Big(\mathsf{RH}_{h,\mathcal{C}}(C_1)\| \times \|\mathsf{RH}_{h,\mathcal{C}}(C_2)\Big), & \text{if } C = C_1 \times C_2; \\ h\Big(\mathsf{RH}_{h,\mathcal{C}}(C_1)\| = \Big), & \text{if } C = (C_1 =) \text{ } (C \text{ ends up with an identity gate}). \end{cases}$$

In the above hash function, we view the gates $+, \times, =$ as binary strings. Recording hash aims to compress the size of the circuits while recording the evaluation. We remark that recording hash is

similar to but different from Merkle hash tree [46]. The major difference is that recording hash also binds the description of each gate of the circuit which is crucial in our construction.

2.3 Indistinguishability and Differing-inputs Obfuscations and Puncturable PRFs.

Indistinguishability obfuscation introduced by Barak *et al.* [12,13] says that obfuscations of two functionally equivalent circuits are indistinguishable. We review the formal definition in Appendix A. A very recent breakthrough work by Garg *et al.*[33] gave the first candidate of indistinguishability obfuscator for all polynomial-size circuits. Other follow-up constructions in [50,11,4,39] were also proposed based on this basic one. A stronger notion called differing-inputs obfuscation was also introduced in [12,13], and many interesting applications are given based on it very recently, *e.g.* [3,14,21]. Differing-inputs obfuscation says that if a PPT algorithm can distinguish the obfuscations of two circuits, then there exists a PPT extractor outputs an input on which the two circuits differ. The formal definition is given in Appendix B. An interesting result given in [21] says that if the circuits only differ in polynomial many inputs, then an indistinguishability obfuscator is already a differing-inputs obfuscator for these circuits. We describe the result in Lemma 2. Puncturable PRFs [51] are special cases of constrained PRFs [20,45,22]. A punctured PRF key associated with a polynomial-size set S is a secret key that allows to evaluate a PRF at all x except $x \in S$. While the values on all the points in S appear pseudorandom to adversaries. We recall the definition of puncturable PRFs in Appendix C.

3 Homomorphic Signatures

Informally, a homomorphic signature scheme [18] consists of the usual $\mathsf{Setup}, \mathsf{Sign}, \mathsf{Verify}$ algorithms as well as an additional evaluation algorithm Eval that translates circuits on messages to circuits on signatures. If $\sigma_1, ..., \sigma_k$ are valid signatures for $m_1, ..., m_k$, then $\mathsf{Eval}((m_1, \sigma_1), ..., (m_k, \sigma_k), C)$ is a valid signature for $C(m_1, ..., m_k)$. We mention that in the definition of [18], the evaluation algorithm only takes the signatures as inputs. Our constructions also need to take the messages as inputs, and this does not affect the functionality of homomorphic schemes. To prevent mixing of data from different data sets when evaluating circuits, $\mathsf{Sign}, \mathsf{Verify}$ and Eval take an additional tag τ as input. The tag serves to bind together messages from the same data set.

A homomorphic signature scheme $\Pi = (\mathsf{Setup}, \mathsf{Sign}, \mathsf{Verify}, \mathsf{Eval})$ is formally defined as follows.

$\mathsf{Setup}(\lambda, k)$: Takes as input the security parameter λ, an integer k indicating the maximal size for a data set. We note that k is given for simplicity, our constructions can be easily extend to variable data set size. It outputs a signing key sk, a verification key vk and an evaluation key ek. Where vk and ek will define a message space $\mathcal{M}$, a signature space Σ and a set $\mathcal{C}$ of admissible circuits $C : \mathcal{M}^k \to \mathcal{M}$.

$\mathsf{Sign}(sk, \tau, m, i)$: Takes as input a signing key sk, a tag $\tau \in \{0,1\}^\lambda$, a message $m \in \mathcal{M}$ and an index $i \in [k]$, outputs a signature $\sigma \in \Sigma$.

$\mathsf{Verify}(vk, \tau, m, \sigma, C)$: Takes as input a verification key vk, a tag $\tau \in \{0,1\}^\lambda$, a message $m \in \mathcal{M}$, a signature $\sigma \in \Sigma$ and a circuit $C \in \mathcal{C}$. It outputs either 1 or 0. We note that the indices $i \in [k]$ are degraded circuits with depth 0 in $\mathcal{C}$.

$\mathsf{Eval}(ek, \tau, \{(m_i, \sigma_i)\}_{i\in[k]}, C)$: Takes as input an evaluation key ek, a tag $\tau \in \{0,1\}^\lambda$, a tuple of message-signature pair $\{(m_i, \sigma_i)\}_{i\in[k]}$ and a circuit $C \in \mathcal{C}$, outputs a signature $\sigma \in \Sigma$.

For correctness, we require that for any $(sk, vk, ek) \leftarrow \mathsf{Setup}(\lambda, k)$, the following holds:

1. For all tags $\tau \in \{0,1\}^\lambda$, all messages $m \in \mathcal{M}$, and all $i \in [k]$, if $\sigma \leftarrow \mathsf{Sign}(sk, \tau, m, i)$ then we have $\mathsf{Verify}(vk, \tau, m, \sigma, i) = 1$.
2. For all tags $\tau \in \{0,1\}^\lambda$, all tuples $(m_1, ..., m_k) \in \mathcal{M}^k$ and all circuits $C \in \mathcal{C}$, if $\sigma_i \leftarrow \mathsf{Sign}(sk, \tau, m_i, i)$, we have $\mathsf{Verify}(vk, \tau, C(m_1, ..., m_k), \mathsf{Eval}(ek, \tau, \{(m_i, \sigma_i)\}_{i\in[k]}, C), C) = 1$.

We say a signature as above is homomorphic for circuits in $\mathcal{C}$ (over $\mathcal{M}$). A signature scheme is *fully homomorphic*, if it is homomorphic for all polynomial-size circuits. A *bounded fully homomorphic* signature scheme is a homomorphic signature scheme where the Setup algorithm takes as additional inputs d, s (now $(sk, vk, ek) \leftarrow \mathsf{Setup}(\lambda, k, d, s)$) and the resulting scheme is homomorphic for circuits in $\mathcal{C}_{d,s}$.

3.1 Unforgeability

Loosely speaking, the unforgeability model for homomorphic signature schemes allows the adversary to make adaptive signature queries on data sets of his choosing as well as a tag τ for the data set he queries. Eventually, the adversary will produce a message-signature pair (m^*, σ^*) as well as a circuit C^* and a tag τ^*. The adversary wins if $(\tau^*, m^*, \sigma^*, C^*)$ passes the validity check and $(\tau^*, m^*) \neq (\tau_j, C^*(m_{j1}, ..., m_{jk}))$ for all queries $\big(\tau_j, (m_{j1}, ..., m_{jk})\big)$. This captures the fact of two types of forgeries. In a type I forgery ($\tau^* \neq \tau_j$ for all j), the successful verification of the tuple (τ^*, m^*, σ^*) indicates that it is a valid signature for some message not queried to the signing oracle. This type corresponds to usual signature forgery. In a type II forgery ($\tau^* = \tau_j$ but $m^* \neq C^*(m_{j1}, ..., m_{jk})$ for some j), the successful verification of the tuple (m^*, σ^*, C^*) indicates that it authenticates m^* as $C^*(m_{j1}, ..., m_{jk})$ but in fact this is not the case. Our unforgeability game is adapted from [18], but with a few differences. That will be compared afterwards.

A homomorphic scheme $\Pi = (\mathsf{Setup}, \mathsf{Sign}, \mathsf{Verify}, \mathsf{Eval})$ is *selectively unforgeable*, if for any PPT algorithm $\mathcal{A}$, the winning probability of $\mathcal{A}$ in the following experiment is negligible.

Sel-UF Experiment:
$(\tau^*, m^*, C^*, \mathsf{st}) \leftarrow \mathcal{A}(\lambda, k)$
$(sk, vk, ek) \leftarrow \mathsf{Setup}(\lambda, k)$
$\sigma^* \leftarrow \mathcal{A}^{\mathsf{Sign}(sk,\cdot,\cdot,\cdot)}(vk, ek, \mathsf{st})$
If $\mathsf{Verify}(vk, \tau^*, m^*, \sigma^*, C^*) = 1$, then $\mathcal{A}$ wins.

The adversary first claims the tag τ^*, the message m^* and the circuit C^* he wants to forge. When making the signature query, $\mathcal{A}$ submits a tag τ and a message m, if m is the first message corresponding to τ, the challenger sets a counter $i_\tau = 1$. Otherwise, the challenger increases the counter i_τ by 1. The challenger answers $\mathcal{A}$ with the signature $\sigma \leftarrow \mathsf{Sign}(sk, \tau, m, i_\tau)$. $\mathcal{A}$ can repeat the interaction at most k times for one τ, but polynomial many tags. A basic requirement here is that the messages associated with the same tag $\mathcal{A}$ asked are subject to $(\tau^*, m^*) \neq (\tau_j, C^*(m_{j1}, ..., m_{jk}))$ for all j, where $(m_{j1}, ..., m_{jk})$ are the messages corresponding to τ_j.

Comparisons of the Models. The unforgeability model defined above are different from the one in [18] as follows. (I) The (adaptive) unforgeability model defined in [18] allows the adversary to output the challenge tag, message and circuit depending on the queries he made. While our model only considers this selective version. We note that one could use standard complexity leveraging method to prove adaptive unforgeability of our schemes. (II) In [18], the adversary is required to submit all messages belonging to one data set at the same time, after which he receives all the signatures on all the messages at once. In our security model, the adversary is allowed to *adaptively* query one message at a time, and even to intersperse queries from different data sets. A similar model was also considered in [32]. (III) In [18], their model requires the tag in the system be unpredictable.

When issuing the signature queries, the tag is chosen by the challenger uniformly at random. While in our model, we allow the adversary to choose the tag on his own choice.

3.2 Privacy

Privacy here means that given signature on a data set $(m_1, ..., m_k)$, the derived signature of the derived message $C(m_1, ..., m_k)$ does not leak any information about $(m_1, ..., m_k)$ beyond what is learned from $C(m_1, ..., m_k)$. More precisely, the definition captures that given signatures on a number of messages derived from two different data sets, the attacker can not tell which data set the derived signatures came from, this holds even the attacker is given the signing key. Ahn *et al.* [2] defined a stronger notion, called *strong context hiding*, that requires the derived signatures be distributed as independent fresh signatures on the same message. Therefore, it even hides the operations on the signatures. Boneh and Freeman [18,19] also gave a weaker privacy notion, called *weak context hiding*, that assumes the original signatures are not public. Our schemes can not hide the fact that derivation took place, then they can not achieve strong context hiding. Interestingly, our schemes ensure the privacy even the original signatures are exposed. More specifically, the derived signatures of our schemes are *identical* as long as the derived messages are the same. We call it *semi-strong context hiding*.

A homomorphic signature $\Pi = (\mathsf{Setup}, \mathsf{Sign}, \mathsf{Verify}, \mathsf{Eval})$ is *semi-strong context hiding*, if for any circuit C (in the admissible circuits), any two tuples of messages $\mathbf{m}_0 = (m_{01}, ..., m_{0k})$, $\mathbf{m}_1 = (m_{11}, ..., m_{1k})$ such that $C(\mathbf{m}_0) = C(\mathbf{m}_1)$, then

$$\mathsf{Eval}(ek, \tau, \{(m_{0i}, \sigma_{0i})\}_{i\in[k]}, C) = \mathsf{Eval}(ek, \tau, \{(m_{1i}, \sigma_{1i})\}_{i\in[k]}, C),$$

for all $\tau \in \{0,1\}^\ell$, $\sigma_{0i} \leftarrow \mathsf{Sign}(sk, \tau, m_{0i}, i)$ and $\sigma_{1i} \leftarrow \mathsf{Sign}(sk, \tau, m_{1i}, i)$ for $i \in [k]$.

Note that the above equality holds even the signing key and the original signatures are exposed. Therefore, this notion of privacy is stronger than weak context hiding.

3.3 Length Efficiency

A homomorphic signature is said *length efficient*, if the length of derived signatures is independent of the size k of the data set. The requirement of length efficiency is to avoid the following trivial solution. The Eval algorithm outputs the circuits C and all the message-signature pairs in the data set. The recipient first checks the validity of each signature and then evaluates C on the original signed messages. Obviously, this construction is not length efficient, since the size of the "derived signature" is linear in the data size. Moreover, it is not context hiding since a "derived signature" reveals everything about the original data set.

4 Constructions

In this section, we give our constructions of bounded fully homomorphic signature schemes. Let the message space $\mathcal{M}$ be a field with *polynomial size*. The circuits over $\mathcal{M}$ contain addition gates $(+)$, product gates $(\times)$ and identity gates $(=)$. Each wire in C connects gates at consecutive level. For polynomials $d = d(\lambda), s = s(\lambda)$, denote $\mathcal{C}_{d,s}$ the circuits over $\mathcal{M}$ with depth and size bounded by d and s, respectively.

Let $i\mathcal{O}$ be an indistinguishability obfuscator for all polynomial size circuits. Our first homomorphic signature scheme, denoted by $\Pi_{d,s} = (\mathsf{Setup}_{d,s}, \mathsf{Sign}_{d,s}, \mathsf{Verify}_{d,s}, \mathsf{Eval}_{d,s})$, for circuits in $\mathcal{C}_{d,s}$ is described as follows.

$\mathsf{Setup}_{d,s}(\lambda, k, d, s)$: It takes as input the security parameter λ, the number of input index $k = k(\lambda)$, the upper bounds d and s of depth and size of circuits. It first generates an injective one-way function $f : \{0,1\}^\lambda \to \{0,1\}^\ell$ with $\ell = \ell(\lambda)$ and a puncturable PRF $\mathsf{F} = (\mathsf{F.Key}, \mathsf{F.Puncture}, \mathsf{F.Eval})$ that maps from $\{0,1\}^\lambda \times \mathcal{M} \times \{0,1\}^s$ to $\{0,1\}^\lambda$. It then generates $d+1$ PRF keys $K_0, ..., K_d \leftarrow \mathsf{F.Key}(1^\lambda)$ and computes the programs $\{\mathsf{Verify}_i\}_{0\le i\le d}$, $\{\mathsf{Eval}_i^+\}_{i\in[d]}$, $\{\mathsf{Eval}_i^\times\}_{i\in[d]}$ and $\{\mathsf{Eval}_i^=\}_{i\in[d]}$ defined in Fig. 1, 2, 3, 4. It outputs the signing key $sk = K_0$, the verification key $vk = \{i\mathcal{O}(\mathsf{Verify}_i)\}_{0\le i\le d}$ and the evaluation key $ek = \{(i\mathcal{O}(\mathsf{Eval}_i^+), i\mathcal{O}(\mathsf{Eval}_i^\times), i\mathcal{O}(\mathsf{Eval}_i^=))\}_{i\in[d]}$. We note that, the algorithm first generates $i\mathcal{O}(\mathsf{Verify}_i)$ for $0 \le i \le d$ and embeds them to the evaluation programs. The message space is $\mathcal{M}$, the signature space is $\{0,1\}^\lambda$ and the admissible circuits set is $\mathcal{C}_{d,s}$.

Verify_i

Constants: f, s, K_i
Input: $c = (\tau, m, \sigma, C)$.
1. If $f(\sigma) = f(\mathsf{F.Eval}(K_i, \tau\|m\|C))$ and $C \in \mathcal{C}_{i,s}$ output 1; otherwise output 0.

Fig. 1.

Eval_i^+

Constants: s, K_i
Input: $c_1 = (\tau_1, m_1, \sigma_1, C_1), c_2 = (\tau_2, m_2, \sigma_2, C_2)$.
1. If $\tau_1 = \tau_2$ continue; otherwise output $\perp$.
2. If $C_1 \in \mathcal{C}_{i-1,s}$ and $C_2 \in \mathcal{C}_{i-1,s}$ continue; otherwise output $\perp$.
3. If $i\mathcal{O}(\mathsf{Verify}_{i-1})(\tau_1, m_1, \sigma_1, C_1) = 1$ and $i\mathcal{O}(\mathsf{Verify}_{i-1})(\tau_2, m_2, \sigma_2, C_2) = 1$ continue; otherwise output $\perp$.
4. Set $\tau = \tau_1 = \tau_2$, $m = m_1 + m_2$ and $C = C_1 + C_2$. // C is the description of circuit .
5. Set $\sigma = \mathsf{F.Eval}(K_i, \tau\|m\|C)$.
6. Output σ.

Fig. 2.

$\mathsf{Eval}_i^\times$

Constants: s, K_i
Input: $c_1 = (\tau_1, m_1, \sigma_1, C_1), c_2 = (\tau_2, m_2, \sigma_2, C_2)$.
1. If $\tau_1 = \tau_2$ continue; otherwise output $\perp$.
2. If $C_1 \in \mathcal{C}_{i-1,s}$ and $C_2 \in \mathcal{C}_{i-1,s}$ continue; otherwise output $\perp$.
3. If $i\mathcal{O}(\mathsf{Verify}_{i-1})(\tau_1, m_1, \sigma_1, C_1) = 1$ and $i\mathcal{O}(\mathsf{Verify}_{i-1})(\tau_2, m_2, \sigma_2, C_2) = 1$ continue; otherwise output $\perp$.
4. Set $\tau = \tau_1 = \tau_2$, $m = m_1 \times m_2$ and $C = C_1 \times C_2$.// C is the description of the circuit.
5. Set $\sigma = \mathsf{F.Eval}(K_i, \tau\|m\|C)$.
6. Output σ.

Fig. 3.

$\mathsf{Sign}_{d,s}(sk, \tau, m, i)$: It takes as input the singing key $sk = K_0$, a tag $\tau \in \{0,1\}^\lambda$, a message $m \in \mathcal{M}$ and an index $i \in [k]$. It then computes $\sigma = \mathsf{F.Eval}(K_0, \tau\|m\|i)$ and outputs σ. The algorithm actually pads i to have length s. For simplicity, we do not explicitly use this padding.

$\mathsf{Eval}_i^{=}$

Constants: s, K_i
Input: $c_1 = (\tau_1, m_1, \sigma_1, C_1)$.

1. If $C_1 \in \mathcal{C}_{i-1,s}$ continue; otherwise output $\perp$.
2. If $i\mathcal{O}(\mathsf{Verify}_{i-1})(\tau_1, m_1, \sigma_1, C_1) = 1$ continue; otherwise output $\perp$.
3. Set $\tau = \tau_1$, $m = m_1$ and $C = (C_1 =)$.// $(C_1 =)$ means C_1 padded with an identity gate.
4. Set $\sigma = \mathsf{F.Eval}(K_i, \tau \| m \| C)$.
5. Output σ.

Fig. 4.

$\mathsf{Verify}_{d,s}(vk, \tau, m, \sigma, C)$: It outputs $i\mathcal{O}(\mathsf{Verify}_{d_C})(\tau, m, \sigma, C)$, where d_C is the depth of C. Note that if C is the input index, then $d_C = 0$.

$\mathsf{Eval}_{d,s}(ek, \tau, \{(m_i, \sigma_i)\}_{i\in[k]}, C)$: For $C \in \mathcal{C}_{d,s}$, the algorithm evaluates the message-signature pairs according to C with inputs $\{(\tau, m_i, \sigma_i, i)\}_{i\in[k]}$ by using $\{i\mathcal{O}(\mathsf{Eval}_i^{+}), i\mathcal{O}(\mathsf{Eval}_i^{\times}), i\mathcal{O}(\mathsf{Eval}_i^{=})\}_{i\in[d]}$ on each level. The algorithm outputs the output σ of the d_C-th level evaluation as the derived signature, where d_C is the depth of C.
We would like to point out that the evaluation algorithm has the data set $(m_1, ..., m_k)$ and the circuit C. Therefore, it could compute all the intermediate messages and subcircuits which will be fed into the evaluation programs in each level. When verifying this derived signature, τ, $C(m_1, ..., m_k)$ and C are additionally provided. Similar consideration was also taken into account in all previous homomorphic signature schemes such as the ones in [18].

4.1 Correctness, Privacy and Length Efficiency

We show the correctness, privacy and length efficiency of our scheme in the following lemma.

Lemma 1. *The homomorphic signature* $\Pi_{d,s} = (\mathsf{Setup}_{d,s}, \mathsf{Sign}_{d,s}, \mathsf{Verify}_{d,s}, \mathsf{Eval}_{d,s})$ *defined above is correct, semi-strong context hiding and length efficient with respect to* $\mathcal{C}_{d,s}$.

Proof. We first prove the correctness. For a fresh signature $\sigma = \mathsf{F.Eval}(K_0, \tau \| m \| i)$, it is easy to verify that $i\mathcal{O}(\mathsf{Verify}_0)(\tau, m, \sigma, i) = 1$. For a derived signature $\sigma \leftarrow \mathsf{Eval}_{d,s}(ek, \tau, \{(m_i, \sigma_i)\}_{i\in[k]}, C)$, where $\sigma_i \leftarrow \mathsf{Sign}_{d,s}(sk, \tau, m_i, i)$ and $C \in \mathcal{C}_{d,s}$. Since the evaluation algorithm evaluates the original message-signature pairs exactly according to C, the derived signature must have the form $\sigma = \mathsf{F.Eval}(K_{d_C}, \tau \| C(m_1, ..., m_k) \| C)$, where d_C is the depth of the circuit C. Thus, one can easily check that $i\mathcal{O}(\mathsf{Verify}_{d_C})(\tau, C(m_1, ..., m_k), \sigma, C) = 1$.

As to the semi-strong context hiding property. For any two tuples of messages $(m_{01}, ..., m_{0k})$ and $(m_{11}, ..., m_{1k})$ and any circuit $C \in \mathcal{C}_{d,s}$ such that $C(m_{01}, ..., m_{0k}) = C(m_{11}, ..., m_{1k}) \triangleq m$. According to the evaluation algorithm, we have that

$$\mathsf{Eval}_{d,s}(ek, \tau, \{m_{0i}, \sigma_{0i}\}_{i\in[k]}, C) = \mathsf{F.Eval}(K_{d_C}, \tau \| m \| C) = \mathsf{Eval}_{d,s}(ek, \tau, \{m_{1i}, \sigma_{1i}\}_{i\in[k]}, C),$$

where $\tau \in \{0,1\}^\lambda$ is arbitrary, $\sigma_{0i} = \mathsf{F.Eval}(K_0, \tau \| m_{0i} \| i)$ and $\sigma_{1i} = \mathsf{F.Eval}(K_0, \tau \| m_{1i} \| i)$ for $i \in [k]$.

The length of fresh and derived signatures in our scheme is λ, which is independent of k. □

4.2 Selective Unforgeability

We first define a series of homomorphic schemes $\Pi_{j,s} = (\mathsf{Setup}_{j,s}, \mathsf{Sign}_{j,s}, \mathsf{Verify}_{j,s}, \mathsf{Eval}_{j,s})$ for circuits in $\mathcal{C}_{j,s}$, where $0 \le j \le d$. Then, we prove the security of $\Pi_{d,s}$ by induction on the depth d. Note that

$\Pi_{0,s}$ is a standard signature scheme, since it does not generate an evaluation key in our construction. We give the following theorems.

Theorem 1. *Assuming $i\mathcal{O}$ is an indistinguishability obfuscator, F is a puncturable PRF and f is a one-way function, then $\Pi_{0,s}$ is selectively unforgeable for circuits in $\mathcal{C}_{0,s}$.*

The proof is essentially the same as the one in [51]. We give it in Appendix D.

Theorem 2. *Assuming $\Pi_{d-1,s}$ is selectively unforgeable for circuits in $\mathcal{C}_{d-1,s}$, $i\mathcal{O}$ is an indistinguishability obfuscator, F is a puncturable PRF and f is an injective one-way function, then $\Pi_{d,s}$ is selectively unforgeable for circuits in $\mathcal{C}_{d,s}$*

We briefly describe the intuition here and the formal proof is given in Appendix E. [3] The proof is shown as a sequence of hybrid experiments where the first experiment is the real selective unforgeability experiment. We prove the winning probabilities of any efficient adversary must be negligibly close between successive experiments. Finally, we show that any efficient adversary only wins with negligible probability in the last experiment.

Let (τ^*, m^*, C^*) be the challenge output by the adversary at the beginning of the experiment. If the depth of C^* is less than d, then it is easy to show that a forgery for $\Pi_{d,s}$ is also a forgery for $\Pi_{d-1,s}$. Hence, we consider the case that the depth of C^* is d. We extend the "punctured program" method of Sahai and Waters [51] and want to remove a key element of the program.

The point $(\tau^* \| m^* \| C^*)$ of the PRF in the depth-d evaluation and verification programs is aimed to be punctured out. We first show how to modify the verification program, which is similar to [51]. The modified verification program we call $\mathsf{Sim.Verify}_d$ is hardwired with constants τ^*, m^*, C^* and $z^* = f(\mathsf{F.Eval}(K_d, \tau^* \| m^* \| C^*))$. It changes the verification step to compare $f(\sigma)$ and z^* when the input contains (τ^*, m^*, C^*) and uses a punctured key $K'_d \leftarrow \mathsf{F.Puncture}(K_d, \{\tau^* \| m^* \| C^*\})$ to check other inputs. It is easy to see that the input-output behaviour of these two programs are identical. By the property of indistinguishability obfuscator, any PPT adversary detects this modification only with negligible probability.

Dealing with the evaluation programs is more challenging. Without loss of generality, we assume C^* ends up with an addition gate $(+)$ and the analysis for other cases are similar. Let us first consider the easier situations. We modify the depth-d evaluation programs corresponding to the product and identity gates to $\mathsf{Sim.Eval}_d^{\times}$ and $\mathsf{Sim.Eval}_d^{=}$. These two modified programs are hardwired with constants τ^*, m^*, C^*. They simply abort if $(\tau, m, C) = (\tau^*, m^*, C^*)$ after the validity check of the input signatures. Otherwise, they use K'_d to re-sign the message and circuit. These modifications do not change the input-output behaviour between the modified and original programs. Notice that the output gate of C^* is an addition gate $(+)$, while these two programs evaluate a product gate $(\times)$ or an identity gate $(=)$. The abort condition above will never be triggered and the evaluations of the punctured PRF under K_d and K'_d are identical for all points except $(\tau^* \| m^* \| C^*)$. Again, by the property of indistinguishability obfuscator, any PPT adversary detects the modifications only with negligible probability.

The modified program $\mathsf{Sim.Eval}_d^{+}$ is similar to $\mathsf{Sim.Eval}_d^{\times}$ and $\mathsf{Sim.Eval}_d^{=}$. However, $\mathsf{Sim.Eval}_d^{+}$ and Eval_d^{+} inherently have differing inputs and we can not directly apply the property of indistinguishability obfuscator. In a bit more details, a differing input $(c_1 = (\tau_1, m_1, \sigma_1, C_1), c_2 = (\tau_2, m_2, \sigma_2, C_2))$ must satisfy that $\tau_1 = \tau_2 = \tau^*$, $C_1 + C_2 = C^*$, $C_1 \in \mathcal{C}_{d-1,s}, C_2 \in \mathcal{C}_{d-1,s}$ and c_1, c_2 pass the verification at depth $(d-1)$. Recall that in the selective unforgeability experiment, all the messages $(m_{j1}, ..., m_{jk})$ associated with some tag τ_j the adversary queried to the signing oracle must satisfy

[3] We note that the proof is easily extended when f has polynomially bounded pre-image size.

that $(\tau^*, m^*) \neq (\tau_j, C^*(m_{j1}, ..., m_{jk}))$. With this, we have that for any messages $(m_{j1}, ..., m_{jk})$ associated with tag τ_j that the adversary queried, must satisfy that $(\tau_1, m_1) \neq (\tau_j, C_1(m_{j1}, ..., m_{jk}))$ or $(\tau_2, m_2) \neq (\tau_j, C_2(m_{j1}, ..., m_{jk}))$. Therefore, a differing input must contain a valid forgery for a circuit in $\mathcal{C}_{d-1,s}$. Furthermore, we show that if the message space size is polynomial and f is injective (or has polynomially bounded pre-image size) then there are only polynomial differing inputs. According to Lemma 2, we conclude that if there exists a PPT adversary distinguishes these two obfuscated programs, then there exists a PPT extractor outputs a differing input, which is a valid forgery for $\Pi_{d-1,s}$.

The rest of the proof is somehow straightforward, we choose $z^* = f(r^*)$ for uniform r^* instead of generating r^* as $r^* = \mathsf{F.Eval}(K_d, \tau^* \| m^* \| C^*)$. This is safe due to the pseudorandomness of PRF. In this case, one can easily show that any PPT adversary can output a forgery only with negligible probability if f is a one-way function.

Remark 1. This scheme supports composition in the sense that outputs of previously derived signatures can be used as inputs for new computations. More specifically, if σ_i is a derived signature for $y_i = C_i(m_1, ..., m_k)$ for $i = 1$ to t, one could compute a derived signature σ for $y = C(y_1, ..., y_t)$ on the inputs $(\sigma_1, .., \sigma_t)$ as long as the depth and size of the circuit $C(C_1, ..., C_t)$ are bounded by d and s, respectively.

Remark 2. The size of the evaluation and verification keys in this scheme is linear in the upper bounds d, s of the circuits depth and size, which might be undesired in some settings. In the next subsection, we extend this scheme to get a construction with succinct evaluation and verification keys whose size only depends on the upper bound d.

4.3 Removing the Restriction on the Circuit Size

In this subsection, we describe our second scheme and remove the restriction on the circuit size. This construction uses the recording hash function for circuits and relies on differing-inputs obfuscation and collision-resistant hash functions.

For any polynomial $d = d(\lambda)$, let $\mathcal{C}_d$ be the set of all circuits over $\mathcal{M}$ with depth bounded by d. Let $h : \{0,1\}^* \to \{0,1\}^\lambda$ be a collision-resistant hash function and $\mathsf{RH}_{h,\mathcal{C}_d}$ be a recording hash function for circuits in $\mathcal{C}_d$ with respect to h. Let $\mathsf{di}\mathcal{O}$ be a differing-inputs obfuscator for all polynomial-size circuits. Our second scheme $\Pi_d = (\mathsf{Setup}_d, \mathsf{Sign}_d, \mathsf{Verify}_d, \mathsf{Eval}_d)$ is described as follows.

$\mathsf{Setup}_d(\lambda, k, d)$: It takes as input the security parameter λ, the number of input index k and the upper bound d of the depth. It first generates a one-way (not necessarily injective) function $f : \{0,1\}^\lambda \to \{0,1\}^\ell$ with $\ell = \ell(\lambda)$ and a puncturable PRF $\mathsf{F} = (\mathsf{F.Key}, \mathsf{F.Puncture}, \mathsf{F.Eval})$ that maps from $\{0,1\}^\lambda \times \mathcal{M} \times \{0,1\}^\lambda$ to $\{0,1\}^\lambda$. It then generates $d+1$ PRF keys $K_0, K_1, ..., K_d \leftarrow \mathsf{F.Key}(1^\lambda)$ and the programs $\{\mathsf{dVerify}_i\}_{0 \le i \le d}$, $\{\mathsf{dEval}_i^+\}_{i \in [d]}$, $\{\mathsf{dEval}_i^\times\}_{i \in [d]}$ and $\{\mathsf{dEval}_i^=\}_{i \in [d]}$ as in Fig 5, 6. Because of the similarity of the evaluation programs, we only describe the programs for the addition gate to save space. It then outputs the signing key $sk = K_0$, the verification key $vk = \{\mathsf{di}\mathcal{O}(\mathsf{dVerify}_i)\}_{0 \le i \le d}$ and the evaluation key $ek = \{(\mathsf{di}\mathcal{O}(\mathsf{dEval}_i^+), \mathsf{di}\mathcal{O}(\mathsf{dEval}_i^\times), \mathsf{di}\mathcal{O}(\mathsf{dEval}_i^=))\}_{i \in [d]}$.

$\mathsf{dVerify}_i$

Constants: f, K_i
Input: $c = (\tau, m, \sigma, t)$.
1. If $f(\sigma) = f(\mathsf{F.Eval}(K_i, \tau \| m \| t))$ output 1; otherwise output 0.

Fig. 5.

dEval_i^+

Constants: K_i, h

Input: $c_1 = (\tau_1, m_1, \sigma_1, t_1), c_2 = (\tau_2, m_2, \sigma_2, t_2)$.

1. If $\tau_1 = \tau_2$ continue; otherwise output $\perp$.
2. If $\mathsf{di}\mathcal{O}(\mathsf{dVerify}_{i-1})(\tau_1, m_1, \sigma_1, t_1) = 1$ and $\mathsf{di}\mathcal{O}(\mathsf{dVerify}_{i-1})(\tau_2, m_2, \sigma_2, t_2) = 1$ continue; otherwise output $\perp$.
3. Set $\tau = \tau_1$, $m = m_1 + m_2$ and $t = h(t_1\| + \|t_2)$.
4. Set $\sigma = \mathsf{F.Eval}(K_i, \tau\|m\|t)$.
5. Output σ.

Fig. 6.

$\mathsf{Sign}_d(sk, \tau, m, i)$: It takes as input the signing key $sk = K_0$, a tag $\tau \in \{0,1\}^\lambda$, a message $m \in \mathcal{M}$ and an index $i \in [k]$. It then computes $t = h(i)$ and outputs $\sigma = \mathsf{F.Eval}(K_0, \tau\|m\|t)$.

$\mathsf{Verify}_d(vk, \tau, m, \sigma, C)$: It first computes $t = \mathsf{RH}_{h,\mathcal{C}_d}(C)$ and then outputs $\mathsf{di}\mathcal{O}(\mathsf{dVerify}_{d_C})$, where d_C is the depth of C.

$\mathsf{Eval}_s(ek, \tau, \{m_i, \sigma_i\}_{i\in[k]}, C)$: The evaluation algorithm is the same as the one in $\Pi_{d,s}$, except that it takes $(\tau, m_i, \sigma_i, h(i))$ as inputs to the evaluation programs for $i \in [k]$.

For correctness, according to the evaluation algorithm, the derived signature of fresh signatures $(\tau, \{m_i, \sigma_i\}_{i\in[k]})$ on circuit $C \in \mathcal{C}_d$ is of the form $\sigma = \mathsf{F.Eval}(\tau\|C(m_1, ..., m_k)\|\mathsf{RH}_{h,\mathcal{C}_d}(C))$. It is easy to see that it will pass the verification.

The proofs of privacy and length efficiency of Π_d easily follow from $\Pi_{d,s}$, and we establish the unforgeability as follows.

Theorem 3. *Assuming* $\mathsf{di}\mathcal{O}$ *is a differing-inputs obfuscator,* f *is a one-way function,* h *is a collision-resistant hash function and* F *is a puncturable PRF, then* Π_d *is selectively unforgeable for circuits in* $\mathcal{C}_d$.

The proof of Theorem 3 is very similar to the proofs of Theorem 1, 2. We only roughly describe it to avoid duplicated work. The difference here is how to modify the evaluation programs. As in the proof of Theorem 2, we consider the evaluation program dEval_d^+.

Given challenge tag τ^*, message m^* and circuit $C^* \in \mathcal{C}_d$. The modified program, denoted by $\mathsf{Sim.dEval}_d^+$, is hardwired with constants $\tau^*, m^*, t^* = \mathsf{RH}_{h,\mathcal{C}_d}(C^*)$. For an input $(c_1 = (\tau_1, m_1, \sigma_1, t_1), c_2 = (\tau_2, m_2, \sigma_2, t_2))$, it runs the first three steps as in dEval_d^+. Then, it aborts if $(\tau, m, h(t_1\| + \|t_2)) = (\tau^*, m^*, t^*)$; otherwise it uses $K'_d \leftarrow \mathsf{F.Puncture}(K_d, \{\tau^*\|m^*\|t^*\})$ to re-sign other inputs. Due to the property of differing-inputs obfuscator, if any PPT adversary can distinguish $\mathsf{di}\mathcal{O}(\mathsf{dEval}_d^+)$ and $\mathsf{di}\mathcal{O}(\mathsf{Sim.dEval}_d^+)$, then there exists a PPT extractor outputs a differing input.

Let $(c_1 = (\tau_1, m_1, \sigma_1, t_1), c_2 = (\tau_2, m_2, \sigma_2, t_2))$ be a differing input, it then must satisfy that $\tau^* = \tau_1 = \tau_2$, $m^* = m_1 + m_2$, $t^* = h(t_1\| + \|t_2)$ and σ_1, σ_2 pass the verification. Denote $C^* \triangleq C_1 + C_2$, we have $t^* = \mathsf{RH}_{h,\mathcal{C}_d}(C^*) = h\big(\mathsf{RH}_{h,\mathcal{C}_d}(C_1)\| + \|\mathsf{RH}_{h,\mathcal{C}_d}(C_2)\big)$ due to the definition of recording hash.

We consider the following cases. (I) If $(t_1, t_2) \neq (\mathsf{RH}_{h,\mathcal{C}_d}(C_1), \mathsf{RH}_{h,\mathcal{C}_d}(C_2))$, then one can efficiently find a collision of the hash function h. (II) If $(t_1, t_2) = (\mathsf{RH}_{h,\mathcal{C}_d}(C_1), \mathsf{RH}_{h,\mathcal{C}_d}(C_2))$, similar analysis shows that c_1, c_2 must contain at least one forgery for circuits C_1, C_2, respectively. Here the number of differing inputs is inherently super-polynomial, even $|\mathcal{M}|$ is polynomial. However, we remark that $|\mathcal{M}|$ still needs to be polynomial to make the proof go through, because in the selective unforgeable experiment, the simulator needs to guess which differing input the extractor will output ahead of time. More details can be seen in Lemma 9 for a similar situation in $\Pi_{d,s}$.

Remark 3. The size of the evaluation and verification keys in Π_d is only linear in the upper bound d of the circuit depth. Π_d still supports composition.

Remark 4. We conjecture that the candidates in [33,50,23,11,39] are differing-inputs obfuscators for all polynomial-size circuits. Garg *et al.* [34] recently showed that the existence of general-purpose differing-inputs obfuscator with auxiliary inputs leads to attacks on some new assumptions they made. We note that no evidence (so far) shows that their attacks on the differing-inputs obfuscator apply to the specific circuits in our construction.

5 Conclusion and Open Problems

In this paper, we propose two bounded fully homomorphic signature schemes, allowing users to publicly verify computations over previous signed data.

Our work leaves several interesting open questions. First, it will be very nice to construct unbounded fully homomorphic signature schemes. A simple try is that one could use only one PRF key in the evaluation key in our second scheme. However, we do not know how to prove its security.

Second, it would be interesting to achieve adaptive unforgeability of our constructions without complexity leveraging. The reason we only achieve selective unforgeability is that we make use of the "punctured program" technique. A promising direction is to adapt the methods in [43], where the authors showed how to achieve adaptively secure signature schemes from indistinguishability obfuscation.

References

1. S. Agrawal and D. Boneh. Homomorphic macs: Mac-based integrity for network coding. In *ACNS*, pages 292–305. Springer, 2009.
2. J. H. Ahn, D. Boneh, J. Camenisch, S. Hohenberger, A. Shelat, and B. Waters. Computing on authenticated data. In *TCC*, pages 1–20. Springer, 2012.
3. P. Ananth, D. Boneh, S. Garg, A. Sahai, and M. Zhandry. Differing-inputs obfuscation and applications. Cryptology ePrint Archive, Report 2013/689, 2013.
4. P. Ananth, D. Gupta, Y. Ishai, and A. Sahai. Optimizing obfuscation: Avoiding barrington's theorem. Cryptology ePrint Archive, Report 2014/222, 2014.
5. B. Applebaum, Y. Ishai, and E. Kushilevitz. From secrecy to soundness: Efficient verification via secure computation. In *ICALP*, pages 152–163. Springer, 2010.
6. G. Ateniese, R. Burns, R. Curtmola, J. Herring, L. Kissner, Z. Peterson, and D. Song. Provable data possession at untrusted stores. In *CCS*, pages 598–609. ACM, 2007.
7. G. Ateniese, S. Kamara, and J. Katz. Proofs of storage from homomorphic identification protocols. In *ASIACRYPT*, pages 319–333. Springer, 2009.
8. N. Attrapadung and B. Libert. Homomorphic network coding signatures in the standard model. In *PKC*, pages 17–34. Springer, 2011.
9. N. Attrapadung, B. Libert, and T. Peters. Computing on authenticated data: New privacy definitions and constructions. In *ASIACRYPT*, pages 367–385. Springer, 2012.
10. N. Attrapadung, B. Libert, and T. Peters. Efficient completely context-hiding quotable and linearly homomorphic signatures. In *PKC*, pages 386–404. Springer, 2013.
11. B. Barak, S. Garg, Y.T. Kalai, O. Paneth, and A. Sahai. Protecting obfuscation against algebraic attacks. In *EUROCRYPT*, pages 221–238. Springer, 2014.
12. B. Barak, O. Goldreich, R. Impagliazzo, S. Rudich, A. Sahai, S. Vadhan, and K. Yang. On the (im) possibility of obfuscating programs. In *CRYPTO*, pages 1–18. Springer, 2001.
13. B. Barak, O. Goldreich, R. Impagliazzo, S. Rudich, A. Sahai, S. Vadhan, and K. Yang. On the (im) possibility of obfuscating programs. *Journal of the ACM (JACM)*, 59(2):6, 2012.
14. M. Bellare and S. Tessaro. Poly-many hardcore bits for any one-way function. Cryptology ePrint Archive, Report 2013/873, 2013.
15. S. Benabbas, R. Gennaro, and Y. Vahlis. Verifiable delegation of computation over large datasets. In *CRYPTO*, pages 111–131. Springer, 2011.

16. N. Bitansky, R. Canetti, A. Chiesa, and E. Tromer. From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again. In *ITCS*, pages 326–349. ACM, 2012.
17. D. Boneh, D. Freeman, J. Katz, and B. Waters. Signing a linear subspace: Signature schemes for network coding. In *PKC*, pages 68–87. Springer, 2009.
18. D. Boneh and D. M. Freeman. Homomorphic signatures for polynomial functions. In *EUROCRYPT*, pages 149–168. Springer, 2011.
19. D. Boneh and D. M. Freeman. Linearly homomorphic signatures over binary fields and new tools for lattice-based signatures. In *PKC*, pages 1–16. Springer, 2011.
20. D. Boneh and B. Waters. Constrained pseudorandom functions and their applications. In *ASIACRYPT*, pages 280–300. Springer, 2013.
21. E. Boyle, K. Chung, and R. Pass. On extractability obfuscation. In *TCC*, pages 52–73. Springer, 2014.
22. E. Boyle, S. Goldwasser, and I. Ivan. Functional signatures and pseudorandom functions. In *PKC*, pages 501–519. Springer, 2014.
23. Z. Brakerski and G.N. Rothblum. Virtual black-box obfuscation for all circuits via generic graded encoding. In *TCC*, pages 1–25. Springer, 2014.
24. D. Catalano and D. Fiore. Practical homomorphic macs for arithmetic circuits. In *EUROCRYPT*, pages 336–352. Springer, 2013.
25. D. Catalano, D. Fiore, R. Gennaro, and L. Nizzardo. Generalizing homomorphic macs for arithmetic circuits. In *PKC*, pages 538–555. Springer, 2014.
26. D. Catalano, D. Fiore, R. Gennaro, and K. Vamvourellis. Algebraic (trapdoor) one-way functions and their applications. In *TCC*, pages 680–699. Springer, 2013.
27. D. Catalano, D. Fiore, and B. Warinschi. Adaptive pseudo-free groups and applications. In *EUROCRYPT*, pages 207–223. Springer, 2011.
28. D. Catalano, D. Fiore, and B. Warinschi. Efficient network coding signatures in the standard model. In *PKC*, pages 680–696. Springer, 2012.
29. K. M. Chung, Y. Kalai, and S. Vadhan. Improved delegation of computation using fully homomorphic encryption. In *CRYPTO*, pages 483–501. Springer, 2010.
30. K. M. Chung, Y. T. Kalai, F. H. Liu, and R. Raz. Memory delegation. In *CRYPTO*, pages 151–168. Springer, 2011.
31. Y. Dodis, S. Vadhan, and D. Wichs. Proofs of retrievability via hardness amplification. In *TCC*, pages 109–127. Springer, 2009.
32. D. M. Freeman. Improved security for linearly homomorphic signatures: A generic framework. In *PKC*, pages 697–714. Springer, 2012.
33. S. Garg, C. Gentry, S. Halevi, M. Raykova, A. Sahai, and B. Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. In *FOCS*, pages 40–49. IEEE, 2013.
34. S. Garg, C. Gentry, S. Halevi, and D. Wichs. On the implausibility of differing-inputs obfuscation and extractable witness encryption with auxiliary input. Cryptology ePrint Archive, Report 2013/860, 2013.
35. R. Gennaro, C. Gentry, and B. Parno. Non-interactive verifiable computing: Outsourcing computation to untrusted workers. In *CRYPTO*, pages 465–482. Springer, 2010.
36. R. Gennaro, J. Katz, H. Krawczyk, and T. Rabin. Secure network coding over the integers. In *PKC*, pages 142–160. Springer, 2010.
37. R. Gennaro and D. Wichs. Fully homomorphic message authenticators. In *ASIACRYPT*, pages 301–320. Springer, 2013.
38. C. Gentry. Fully homomorphic encryption using ideal lattices. In *STOC*, pages 169–178, 2009.
39. C. Gentry, A. Lewko, A. Sahai, and B. Waters. Indistinguishability obfuscation from the multilinear subgroup elimination assumption. Cryptology ePrint Archive, Report 2014/309, 2014.
40. C. Gentry and D. Wichs. Separating succinct non-interactive arguments from all falsifiable assumptions. In *STOC*, pages 99–108. ACM, 2011.
41. O. Goldreich, S. Goldwasser, and S. Micali. How to construct random functions (extended abstract). In *FOCS*, pages 464–479, 1984.
42. S. Goldwasser, Y. T. Kalai, and G. N. Rothblum. Delegating computation: interactive proofs for muggles. In *STOC*, pages 113–122. ACM, 2008.
43. S. Hohenberger, A. Sahai, and B. Waters. Replacing a random oracle: Full domain hash from indistinguishability obfuscation. In *EUROCRYPT*, pages 201–220. Springer, 2014.
44. R. Johnson, D. Molnar, D. Song, and D. Wagner. Homomorphic signature schemes. In *CT-RSA*, pages 244–262. Springer, 2002.
45. A. Kiayias, S. Papadopoulos, N. Triandopoulos, and T. Zacharias. Delegatable pseudorandom functions and applications. In *CCS*, pages 669–684. ACM, 2013.
46. R. C. Merkle. A digital signature based on a conventional encryption function. In *CRYPTO*, pages 369–378. Springer, 1988.

47. S. Micali. Computationally sound proofs. *SIAM Journal on Computing*, 30(4):1253–1298, 2000.
48. C. Papamanthou, E. Shi, and R. Tamassia. Signatures of correct computation. In *TCC*, pages 222–242. Springer, 2013.
49. B. Parno, M. Raykova, and V. Vaikuntanathan. How to delegate and verify in public: Verifiable computation from attribute-based encryption. In *TCC*, pages 422–439. Springer, 2012.
50. R. Pass, K. Seth, and S. Telang. Indistinguishability obfuscation from semantically-secure multilinear encodings. Cryptology ePrint Archive, Report 2013/781, 2013.
51. A. Sahai and B. Waters. How to use indistinguishability obfuscation: Deniable encryption, and more. In *STOC*, pages ??–?? ACM, 2014.
52. H. Shacham and B. Waters. Compact proofs of retrievability. In *ASIACRYPT*, pages 90–107. Springer, 2008.
53. A. Shpilka and A. Yehudayoff. Arithmetic circuits: A survey of recent results and open questions. *Foundations and Trends in Theoretical Computer Science*, 5(3–4):207–388, 2010.

A Indistinguishability Obfuscation

The notion of indistinguishability obfuscation was first introduced by Barak *et al.* [12,13]. A uniform PPT machine $i\mathcal{O}$ is called an indistinguishability obfuscator for a circuit class $\{\mathcal{C}_\lambda\}_{\lambda\in\mathbb{N}}$ if the following holds:

1. For all $\lambda \in \mathbb{N}$, every $C \in \mathcal{C}_\lambda$, every input x in the domain of C, we have that
$$\Pr[\tilde{C} \leftarrow i\mathcal{O}(C) : \tilde{C}(x) = C(x)] = 1.$$
2. For any PPT algorithm $\mathcal{D}$, any two circuits $C_0, C_1 \in \mathcal{C}_\lambda$, if $C_0(x) = C_1(x)$ for all input x, then we have
$$|\Pr[\mathcal{D}(i\mathcal{O}(C_0)) = 1] - \Pr[\mathcal{D}(i\mathcal{O}(C_1)) = 1]| \leq \mathrm{negl}(\lambda).$$

A recent breakthrough work of Garg *et al.* [33] first shows how to get a candidate construction of indistinguishability obfuscators for all polynomial-size circuits under novel algebraic hardness assumptions.

B Differing-Inputs Obfuscation

A stronger notion called differing-inputs obfuscation or extractability obfuscation was also introduced in [12,13]. Very recently, many interesting applications were given based on it [3,14,21]. A uniform PPT machine $\mathsf{di}\mathcal{O}$ is called a differing-inputs obfuscator for a circuit class $\{\mathcal{C}_\lambda\}_{\lambda\in\mathbb{N}}$ if the following holds.

1. For all $\lambda \in \mathbb{N}$, every $C \in \mathcal{C}_\lambda$, every input x in the domain of C, we have that
$$\Pr[\tilde{C} \leftarrow \mathsf{di}\mathcal{O}(C) : \tilde{C}(x) = C(x)] = 1.$$
2. For any PPT distinguisher $\mathcal{D}$ and polynomial $p(\lambda)$, there exists a PPT extractor E and polynomial $q(\lambda)$ such that the following holds. For all $\lambda \in \mathbb{N}$, any two circuits $C_0, C_1 \in \mathcal{C}_\lambda$ and any auxiliary input z,
$$|\Pr[\tilde{C} \leftarrow \mathsf{di}\mathcal{O}(C_0) : \mathcal{D}(\tilde{C}, C_0, C_1, z) = 1] - \Pr[\tilde{C} \leftarrow \mathsf{di}\mathcal{O}(C_1) : \mathcal{D}(\tilde{C}, C_0, C_1, z) = 1]| \geq \frac{1}{p(\lambda)}$$
$$\Longrightarrow \quad \Pr[x \leftarrow \mathsf{E}(C_0, C_1, z) : C_0(x) \neq C_1(x)] \geq \frac{1}{q(\lambda)}.$$

An interesting result given in [21] says that if any two circuits differ on at most polynomial many inputs, then an indistinguishability obfuscator on those circuits is also a differing-inputs obfuscator. More specifically, we describe the following lemma.

Lemma 2 ([21] Lemma 6.3). *Let $N = N(\lambda)$ be a polynomial, $i\mathcal{O}$ be an indistinguishability obfuscator. For any distinguisher $\mathcal{D}$ and polynomial $p(\lambda)$, there exists a* PPT *extractor* E *and polynomial $t_{\mathsf{E}}(\lambda, N)$ such that the following holds. For any two circuits C_0, C_1 differ on at most N inputs and any auxiliary input z,*

$$|\Pr[\tilde{C} \leftarrow i\mathcal{O}(C_0) : \mathcal{D}(\tilde{C}, C_0, C_1, z) = 1] - \Pr[\tilde{C} \leftarrow i\mathcal{O}(C_1) : \mathcal{D}(\tilde{C}, C_0, C_1, z) = 1]| \geq \frac{1}{p(\lambda)}$$
$$\implies \quad \Pr[x \leftarrow \mathsf{E}(C_0, C_1, z) : C_0(x) \neq C_1(x)] \geq 1 - \mathit{negl}(\lambda)$$

and E *terminates within time $t_{\mathsf{E}}(\lambda, N)$.*

All the applications based on differing-inputs obfuscation proposed before conjecture that differing-inputs obfuscator for polynomial-size circuits exists by assuming the candidate in [33] is a differing-inputs obfuscator or assuming the candidates satisfy a stronger security, *virtual black-box security*, in an ideal model [23,11] are differing-inputs obfuscators for NC^1 circuits and amplifying them to obfuscate polynomial-size circuits by fully homomorphic encryption. Garg *et al* [34] showed that the existence of general-purpose differing-inputs obfuscator with auxiliary inputs leads to attacks on some new assumptions they made. Although the assumptions may be false, it reminds us to be cautious to general-purpose differing-inputs obfuscators with auxiliary inputs.

C Puncturable Pseudorandom Functions

We recall the definition of puncturable PRF given in [51]. A puncturable family of PRFs consists of three algorithms $\mathsf{F} = (\mathsf{F.Key}, \mathsf{F.Puncture}, \mathsf{F.Eval})$:

- **Key Generation** $\mathsf{F.Key}(1^\lambda)$ takes as input the security parameter λ and outputs a PRF key K.
- **Punctured Key Generation** $\mathsf{F.Puncture}(K, S)$ takes as input a PRF key K, a set $S \subset \{0,1\}^\lambda$ and outputs a punctured key K_S.
- **Evaluation** $\mathsf{F.Eval}(K, x)$ is a deterministic algorithm that takes as input a key K (punctured key or PRF key), a string $x \in \{0,1\}^\lambda$ and outputs y.

A family of PRFs $\mathsf{F} = (\mathsf{F.Key}, \mathsf{F.Puncture}, \mathsf{F.Eval})$ is puncturable if it satisfies the following properties:

- **Functionality preserved under puncturing.** Let $K \leftarrow \mathsf{F.Key}(1^\lambda)$, $K_S \leftarrow \mathsf{F.Puncture}(K, S)$. Then, for all $x \notin S$, $\mathsf{F.Eval}(K, x) = \mathsf{F.Eval}(K_S, x)$.
- **Pseudorandom at punctured points.** For every adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ such that $\mathcal{A}_1(\lambda)$ outputs a set $S \subset \{0,1\}^\lambda$ and $x \in S$, for any $K \leftarrow \mathsf{F.Key}(1^\lambda)$ and $K_S \leftarrow \mathsf{F.Puncture}(K, S)$, the following holds.

$$|\Pr[\mathcal{A}_2(K_S, x, \mathsf{F.Eval}(K, x)) = 1] - \Pr[\mathcal{A}_2(K_S, x, U) = 1]| \leq \mathrm{negl}(\lambda)$$

  where U denotes the uniform distribution.

The GGM tree-based construction of PRFs [41] from one-way functions are easily seen to yield puncturable PRFs, as recently observed by [20,45,22].

D Proof of Theorem 1

Before giving the proof, we note that the $\Pi_{0,s}$ scheme is actually a standard signature scheme. It does not involve an evaluation algorithm. More specifically, $\Pi_{0,s} = (\mathsf{Setup}_{0,s}, \mathsf{Sign}_{0,s}, \mathsf{Verify}_{0,s})$, where the signing key is $sk = K_0$, the verification key is $vk = i\mathcal{O}(\mathsf{Verify}_0)$.

We give the proof as a sequence of hybrid experiments where the first one corresponds to the original experiment of the $\Pi_{0,s}$ scheme. We prove that the winning probabilities of any PPT adversary must be negligibly close between successive experiments. Then, we show that any PPT adversary in the final experiment wins only with negligible probability.

$\mathbf{Hyb}_0$: This experiment is the same as the selective unforgeability experiment. More specifically, it does the following.

1. The adversary $\mathcal{A}$ outputs the challenge tag $\tau^* \in \{0,1\}^\lambda$, challenge message $m^* \in \mathcal{M}$ and the challenge index $i^* \in [k]$ he wants to forge.
2. The challenger runs the $\mathsf{Setup}_{0,s}(\lambda, k, 0, s)$ algorithm and gives $vk = i\mathcal{O}(\mathsf{Verify}_0)$ to $\mathcal{A}$ and keeps the signing key $sk = K_0$.
3. $\mathcal{A}$ submits a query (τ, m) to the signing oracle. If m is the first message corresponding to τ, the challenger sets a counter $i_\tau = 1$. Otherwise, the challenger increases the counter i_τ by 1. The challenger responds to $\mathcal{A}$ with the signature $\sigma = \mathsf{F.Eval}(K_0, \tau \| m \| i_\tau)$. $\mathcal{A}$ can query the oracle at most k times for one τ (but for polynomial many tags). Let $\{m_{ji}\}_{i\in[k]}$ be the messages $\mathcal{A}$ queried corresponding to τ_j, they must satisfy $(\tau^*, m^*) \neq (\tau_j, m_{j(i^*)})$ for all j.
4. $\mathcal{A}$ outputs a forgery σ^*. If $i\mathcal{O}(\mathsf{Verify}_0)(\tau^*, m^*, \sigma^*, i^*) = 1$, then $\mathcal{A}$ wins. Let X_0 be the event that $\mathcal{A}$ wins.

$\mathbf{Hyb}_1$: This experiment is the same as $\mathbf{Hyb}_0$ except the challenger sets $z^* = f(\mathsf{F.Eval}(K_0, \tau^* \| m^* \| i^*))$, generates $K_0' \leftarrow \mathsf{F.Puncture}(K_0, \{\tau^* \| m^* \| i^*\})$ and modifies the Verify_0 program to $\mathsf{Sim.Verify}_0$, where $\mathsf{Sim.Verify}_0$ is defined in Fig. 7. Let X_1 be the event that $\mathcal{A}$ wins.

$\mathsf{Sim.Verify}_0$

Constants: $f, s, K_0', \tau^*, m^*, i^*, z^*$
Input: $c = (\tau, m, \sigma, i)$.

1. If $(\tau, m, i) = (\tau^*, m^*, i^*)$, test if $f(\sigma) = z^*$. If it is true, output 1; otherwise output 0.
2. Else, test if $f(\sigma) = f(\mathsf{F.Eval}(K_0', \tau \| m \| i^*))$. If it is true, output 1; otherwise output 0.

Fig. 7.

$\mathbf{Hyb_2}$: This experiment is the same as $\mathbf{Hyb}_1$, except that the challenger chooses $r^* \leftarrow \{0,1\}^\lambda$ uniformly at random and sets $z^* = f(r^*)$. Let X_2 be the event that $\mathcal{A}$ wins.

Lemma 3. *Assuming $i\mathcal{O}$ is an indistinguishability obfuscator, then* $\Pr[X_0] \leq \Pr[X_1] + negl(\lambda)$.

Proof. We note that the only difference between the two experiments is that $\mathbf{Hyb}_0$ uses $i\mathcal{O}(\mathsf{Verify}_0)$ as the verification key while $\mathbf{Hyb}_1$ uses $i\mathcal{O}(\mathsf{Sim.Verify}_0)$. It is also easy to see that Verify_0 and $\mathsf{Sim.Verify}_0$ have the same input-output behaviour. Suppose there exists a PPT adversary $\mathcal{A}$ such that $\Pr[X_0] - \Pr[X_1] \geq 1/\mathrm{poly}(\lambda)$, we now construct an algorithm $\mathcal{B}$ to break the indistinguishability obfuscator.

Receiving (τ^*, m^*, i^*) from $\mathcal{A}$, $\mathcal{B}$ generates the one-way function f, the signing key $K_0 \leftarrow \mathsf{F.Key}(1^\lambda)$ and the punctured key $K_0' \leftarrow \mathsf{F.Puncture}(K_0, \{\tau^* \| m^* \| i^*\})$. $\mathcal{B}$ then creates the programs Verify_0 and $\mathsf{Sim.Verify}_0$ and submits them to his $i\mathcal{O}$ challenger and receives $i\mathcal{O}(\mathsf{Verify})$. $\mathcal{B}$ sets $vk = i\mathcal{O}(\mathsf{Verify})$ and sends it to $\mathcal{A}$, and answers $\mathcal{A}$'s signing queries with K_0. $\mathcal{B}$ outputs 1 if $\mathcal{A}$ wins. Note that if $\mathsf{Verify} = \mathsf{Verify}_0$, then $\mathcal{A}$ is in $\mathbf{Hyb}_0$; if $\mathsf{Verify} = \mathsf{Sim.Verify}_0$, then $\mathcal{A}$ is in $\mathbf{Hyb}_1$. □

Lemma 4. *Assuming* F *is a puncturable PRF, then* $\Pr[X_1] \leq \Pr[X_2] + negl(\lambda)$.

Proof. Suppose there exists a PPT adversary $\mathcal{A}$ such that $\Pr[X_1] - \Pr[X_2] \geq 1/\text{poly}(\lambda)$, we construct a PPT algorithm $\mathcal{B}$ to break the pseudorandomness of the puncturable PRF. Receiving (τ^*, m^*, i^*) from $\mathcal{A}$, $\mathcal{B}$ submits the punctured set $\{\tau^* \| m^* \| i^*\}$ to his PRF challenger. $\mathcal{B}$ gets K' and r, where $K' \leftarrow \mathsf{F.Punture}(K, \{\tau^* \| m^* \| i^*\})$ and either $r = \mathsf{F.Eval}(K, \tau^* \| m^* \| i^*)$ or r is uniformly random from $\{0,1\}^\lambda$. $\mathcal{B}$ generates f and creates $\mathsf{Sim.Verify}_0$ by setting $K'_0 = K'$ and $z^* = f(r)$. $\mathcal{B}$ then sends $i\mathcal{O}(\mathsf{Sim.Verify}_0)$ to $\mathcal{A}$ and answers $\mathcal{A}$'s signing queries using K'. Note that $\mathcal{A}$ will never query (τ^*, m^*, i^*), then $\mathcal{B}$ will answer correctly. Finally, $\mathcal{B}$ outputs 1 if $\mathcal{A}$ wins. If $r = \mathsf{F.Eval}(K, \tau^* \| m^* \| i^*)$, then $\mathcal{A}$ is in $\mathbf{Hyb}_1$, if $r \leftarrow \{0,1\}^\ell$, then $\mathcal{A}$ is in $\mathbf{Hyb}_2$. □

Lemma 5. *Assuming* f *is a one-way function, then* $\Pr[X_2] \leq negl(\lambda)$.

Proof. Let $\mathcal{A}$ be a PPT adversary that outputs a valid forgery in $\mathbf{Hyb}_2$, we construct a PPT algorithm $\mathcal{B}$ to break the one-wayness of f. Receiving $z^* = f(r^*)$ with $r^* \leftarrow \{0,1\}^\ell$ from the one-way function challenger, $\mathcal{B}$ generates the signing key $sk = K_0$ and the verification key vk by using f, z^* and the punctured key $K'_0 \leftarrow \mathsf{F.Puncture}(K_0, \{\tau^* \| m^* \| i^*\})$. $\mathcal{B}$ answers the signing queries from $\mathcal{A}$ with sk and outputs whatever $\mathcal{A}$ outputs. Note that if $\mathcal{A}$ outputs a valid forgery σ^*, it must satisfy $\mathsf{Sim.Verify}_0(\tau^*, m^*, i^*) = 1$. Therefore, we have $f(\sigma^*) = z^*$, and σ^* is a pre-image of z^*. □

E Proof of Theorem 2

Proof. Let (τ^*, m^*, C^*) be the challenge output by the adversary at the beginning of the experiment. We consider two cases as follows.

Case 1: If the depth of C^* is less than d, denoted by $d_{C^*} \leq d-1$. We show that $\Pi_{d,s}$ is selectively unforgeable for circuits in $\mathcal{C}_{d,s}$, assuming $\Pi_{d-1,s}$ is selectively unforgeable for circuits in $\mathcal{C}_{d-1,s}$. Let $\mathcal{A}$ be a PPT adversary attacks $\Pi_{d,s}$, we construct a PPT adversary $\mathcal{B}$ to attack $\Pi_{d-1,s}$. Receiving (τ^*, m^*, C^*) from $\mathcal{A}$, $\mathcal{B}$ outputs (τ^*, m^*, C^*) as the challenge to the scheme $\Pi_{d-1,s}$. Then $\mathcal{B}$ receives the verification key $vk' = \{i\mathcal{O}(\mathsf{Verify}_i)\}_{0 \leq i \leq d-1}$ and the evaluation key $ek' = \{(i\mathcal{O}(\mathsf{Eval}_i^+), i\mathcal{O}(\mathsf{Eval}_i^\times), i\mathcal{O}(\mathsf{Eval}_i^=))\}_{i \in [d-1]}$. $\mathcal{B}$ then creates the verification program Verify_d and the evaluation programs Eval_d^+, $\mathsf{Eval}_d^\times$ and $\mathsf{Eval}_d^=$ by generating a fresh PRF key $K_d \leftarrow \mathsf{F.Key}(1^\lambda)$ [4]. $\mathcal{B}$ sets $ek = (ek', i\mathcal{O}(\mathsf{Eval}_d^+), i\mathcal{O}(\mathsf{Eval}_d^\times), i\mathcal{O}(\mathsf{Eval}_d^=))$ and $vk = (vk', i\mathcal{O}(\mathsf{Verify}_d))$ and sends ek, vk to $\mathcal{A}$. $\mathcal{B}$ answers $\mathcal{A}$'s queries using his own signing oracle. Notice that $\mathcal{B}$ will always answer $\mathcal{A}$'s queries correctly. $\mathcal{B}$ outputs whatever $\mathcal{A}$ outputs. It is easy to see that if $\mathcal{A}$ outputs a valid forgery with non-negligible probability, then $\mathcal{B}$ will outputs a valid forgery with the same probability.

Case 2: If the depth of C^* is d. We give the proof as a sequence of hybrid experiments where the first one corresponds to the original experiment of the $\Pi_{d,s}$ scheme. We prove that the winning probabilities of any PPT adversary are negligibly close between successive experiments. Finally, we show that any PPT adversary in the final experiment wins only with negligible probability.

$\mathbf{Hyb}_0$: This experiment is the same as the selective unforgeability experiment. More specifically, it does the following.

1. The adversary $\mathcal{A}$ outputs the challenge tag $\tau^* \in \{0,1\}^\lambda$, challenge message $m^* \in \mathcal{M}$ and challenge circuit $C^* \in \mathcal{C}_{d,s}$ he wants to forge. Without loss of generality, we assume C^* ends up with an addition gate $(+)$, and we can prove the theorem in a similar way for other cases.
2. The challenger runs the $\mathsf{Setup}_{d,s}(\lambda, k, d, s)$ algorithm and gives $vk = \{i\mathcal{O}(\mathsf{Verify}_i)\}_{0 \leq i \leq d}$ and $ek = \{(i\mathcal{O}(\mathsf{Eval}_i^+), i\mathcal{O}(\mathsf{Eval}_i^\times), i\mathcal{O}(\mathsf{Eval}_i^=))\}_{i \in [d]}$ to $\mathcal{A}$ and keeps the signing key $sk = K_0$.

[4] The one-way function f is implicitly given in the verification key.

3. $\mathcal{A}$ submits a query (τ, m) to the signing oracle. If m is the first message corresponding to τ, the challenger sets a counter $i_\tau = 1$. Otherwise, the challenger increases the counter i_τ by 1. The challenger answers $\mathcal{A}$ with signature $\sigma = \mathsf{F.Eval}(K_0, \tau \| m \| i_\tau)$. $\mathcal{A}$ can query the oracle at most k times for one τ (but for polynomial many tags). Let $\{m_{ji}\}_{i\in[k]}$ be the messages $\mathcal{A}$ queried corresponding to τ_j, they must satisfy $(\tau^*, m^*) \neq (\tau_j, C^*(m_{j1}, ..., m_{jk}))$ for all j.
4. $\mathcal{A}$ outputs a forgery σ^*. If $i\mathcal{O}(\mathsf{Verify}_d)(\tau^*, m^*, \sigma^*, C^*) = 1$, then $\mathcal{A}$ wins. Let X_0^d be the event that $\mathcal{A}$ wins.

$\mathbf{Hyb}_1$: This experiment is the same as $\mathbf{Hyb}_0$ except the challenger sets $z^* = f(\mathsf{F.Eval}(K_d, \tau^* \| m^* \| C^*))$, generates $K'_d \leftarrow \mathsf{F.Puncture}(K_d, \{\tau^* \| m^* \| C^*\})$ and modifies the Verify_d program to $\mathsf{Sim.Verify}_d$, where $\mathsf{Sim.Verify}_d$ is defined in Fig. 8. Let X_1^d be the event that $\mathcal{A}$ wins.

$\mathsf{Sim.Verify}_d$

Constants: $f, s, K'_d, \tau^*, m^*, C^*, z^*$
Input: $c = (\tau, m, \sigma, C)$.

1. If $(\tau, m, C) = (\tau^*, m^*, C^*)$, test if $f(\sigma) = z^*$ and $C \in \mathcal{C}_{d,s}$. If it is true, output 1; otherwise output 0.
2. Else, test if $f(\sigma) = f(\mathsf{F.Eval}(K'_d, \tau \| m \| C))$ and $C \in \mathcal{C}_{d,s}$. If it is true, output 1; otherwise output 0.

Fig. 8.

$\mathbf{Hyb}_2$: This experiment is the same as $\mathbf{Hyb}_1$ except the challenger modifies $\mathsf{Eval}_d^\times$ to $\mathsf{Sim.Eval}_d^\times$, where $\mathsf{Sim.Eval}_d^\times$ is defined in Fig. 9 and $K'_d \leftarrow \mathsf{F.Puncture}(K_d, \{\tau^* \| m^* \| C^*\})$ is the punctured key as in $\mathbf{Hyb}_1$. Let X_2^d be the event that $\mathcal{A}$ wins.

$\mathsf{Sim.Eval}_d^\times$

Constants: $s, K'_d, \tau^*, m^*, C^*$
Input: $c_1 = (\tau_1, m_1, \sigma_1, C_1), c_2 = (\tau_2, m_2, \sigma_2, C_2)$.

1. If $\tau_1 = \tau_2$ continue; otherwise output $\perp$.
2. If $C_1 \in \mathcal{C}_{d-1,s}$ and $C_2 \in \mathcal{C}_{d-1,s}$ continue; otherwise output $\perp$.
3. If $i\mathcal{O}(\mathsf{Verify}_{d-1})(\tau_1, m_1, \sigma_1, C_1) = 1$ and $i\mathcal{O}(\mathsf{Verify}_{d-1})(\tau_2, m_2, \sigma_2, C_2) = 1$ continue; otherwise output $\perp$.
4. Set $\tau = \tau_1 = \tau_2$, $m = m_1 \times m_2$ and $C = C_1 \times C_2$.
5. If $(\tau, m, C) = (\tau^*, m^*, C^*)$ output $\perp$.
6. Set $\sigma = \mathsf{F.Eval}(K'_d, \tau \| m \| C)$.
7. Output σ.

Fig. 9.

$\mathbf{Hyb}_3$: This experiment is the same as $\mathbf{Hyb}_2$ except the challenger modifies $\mathsf{Eval}_d^=$ to $\mathsf{Sim.Eval}_d^=$, where $\mathsf{Sim.Eval}_d^=$ is defined in Fig. 10 and $K'_d \leftarrow \mathsf{F.Punture}(K_d, \{\tau^* \| m^* \| C^*\})$ is the punctured key as in $\mathbf{Hyb}_2$. Let X_3^d be the event that $\mathcal{A}$ wins.

$\mathbf{Hyb}_4$: This experiment is the same as $\mathbf{Hyb}_3$ except the challenger modifies the program Eval_d^+ to $\mathsf{Sim.Eval}_d^+$, where $\mathsf{Sim.Eval}_d^+$ is defined in Fig. 11 and $K'_d \leftarrow \mathsf{F.Punture}(K_d, \{\tau^* \| m^* \| C^*\})$ is the punctured key as in $\mathbf{Hyb}_3$. Let X_4^d be the event that $\mathcal{A}$ wins.

$\mathsf{Sim.Eval}_d^{=}$

Constants: $s, K'_d, \tau^*, m^*, C^*$
Input: $c_1 = (\tau_1, m_1, \sigma_1, C_1)$.

1. If $C_1 \in \mathcal{C}_{d-1,s}$ continue; otherwise output $\perp$.
2. If $i\mathcal{O}(\mathsf{Verify}_{d-1})(\tau_1, m_1, \sigma_1, C_1) = 1$ continue; otherwise output $\perp$.
3. Set $\tau = \tau_1$, $m = m_1$ and $C = (C_1 =)$.
4. If $(\tau, m, C) = (\tau^*, m^*, C^*)$ output $\perp$.
5. Set $\sigma = \mathsf{F.Eval}(K'_d, \tau \| m \| C)$.
6. Output σ.

Fig. 10.

$\mathsf{Sim.Eval}_d^{+}$

Constants: $s, K'_d, \tau^*, m^*, C^*$
Input: $c_1 = (\tau_1, m_1, \sigma_1, C_1), c_2 = (\tau_2, m_2, \sigma_2, C_2)$.

1. If $\tau_1 = \tau_2$ continue; otherwise output $\perp$.
2. If $C_1 \in \mathcal{C}_{d-1,s}$ and $C_2 \in \mathcal{C}_{d-1,s}$ continue; otherwise output $\perp$.
3. If $i\mathcal{O}(\mathsf{Verify}_{d-1})(\tau_1, m_1, \sigma_1, C_1) = 1$ and $i\mathcal{O}(\mathsf{Verify}_{d-1})(\tau_2, m_2, \sigma_2, C_2) = 1$ continue; otherwise output $\perp$.
4. Set $\tau = \tau_1 = \tau_2$, $m = m_1 + m_2$ and $C = C_1 + C_2$.
5. If $(\tau, m, C) = (\tau^*, m^*, C^*)$ output $\perp$.
6. Set $\sigma = \mathsf{F.Eval}(K'_d, \tau \| m \| C)$.
7. Output σ.

Fig. 11.

$\mathbf{Hyb_5}$: This experiment is the same as $\mathbf{Hyb}_4$ except the challenger chooses $r^* \leftarrow \{0,1\}^\lambda$ uniformly at random and sets $z^* = f(r^*)$. Let X_5^d be the event that $\mathcal{A}$ wins.

Lemma 6. *Assuming $i\mathcal{O}$ is an indistinguishability obfuscator, then* $\Pr[X_0^d] \leq \Pr[X_1^d] + negl(\lambda)$.

Proof. We note that the only difference between the two experiments is that $\mathbf{Hyb}_0$ includes $i\mathcal{O}(\mathsf{Verify}_d)$ in the verification key while $\mathbf{Hyb}_1$ includes $i\mathcal{O}(\mathsf{Sim.Verify}_d)$. It is also easy to see that Verify_d and $\mathsf{Sim.Verify}_d$ have the same input-output behaviour. Suppose there exists a PPT adversary $\mathcal{A}$ such that $\Pr[X_0^d] - \Pr[X_1^d] \geq 1/\text{poly}(\lambda)$, we now construct an algorithm $\mathcal{B}$ to break the indistinguishability obfuscator.

Receiving (τ^*, m^*, C^*) from $\mathcal{A}$, $\mathcal{B}$ creates the programs Verify_d and $\mathsf{Sim.Verify}_d$. $\mathcal{B}$ then submits them to his $i\mathcal{O}$ challenger and receives $i\mathcal{O}(\mathsf{Verify})$. Note that $\mathcal{B}$ can first generate the one-way function f, generate a fresh key $K_d \leftarrow \mathsf{F.Key}(1^\lambda)$, and then compute the punctured key $K'_d \leftarrow \mathsf{F.Puncture}(K_d, \{\tau^* \| m^* \| C^*\})$. $\mathcal{B}$ then generates the signing key $sk = K_0$, the verification programs $\{\mathsf{Verify}_i\}_{0 \leq i \leq d-1}$ and the evaluation programs $\{(\mathsf{Eval}_i^+, \mathsf{Eval}_i^\times, \mathsf{Eval}_i^=)\}_{i \in [d]}$. $\mathcal{B}$ then sets the evaluation key $ek = \{(i\mathcal{O}(\mathsf{Eval}_i^+), i\mathcal{O}(\mathsf{Eval}_i^\times), i\mathcal{O}(\mathsf{Eval}_i^=))\}_{i \in [d]}$ and the verification key $vk = (\{i\mathcal{O}(\mathsf{Verify}_i)\}_{0 \leq i \leq d-1}, i\mathcal{O}(\mathsf{Verify}))$. $\mathcal{B}$ sends ek, vk to $\mathcal{A}$ and answers $\mathcal{A}$'s signing queries with K_0. $\mathcal{B}$ outputs 1 if $\mathcal{A}$ wins. Note that if $\mathsf{Verify} = \mathsf{Verify}_d$, then $\mathcal{A}$ is in $\mathbf{Hyb}_0$; if $\mathsf{Verify} = \mathsf{Sim.Verify}_d$, then $\mathcal{A}$ is in $\mathbf{Hyb}_1$. □

Lemma 7. *Assuming $i\mathcal{O}$ is an indistinguishability obfuscator, then* $\Pr[X_1^d] \leq \Pr[X_2^d] + negl(\lambda)$.

Proof. The only difference between $\mathbf{Hyb}_1$ and $\mathbf{Hyb}_2$ is the difference between the programs $\mathsf{Eval}_d^\times$ and $\mathsf{Sim.Eval}_d^\times$. K_d is used to re-sign the product of the two messages in $\mathsf{Eval}_d^\times$. While K'_d is used in $\mathsf{Sim.Eval}_d^\times$, where K'_d is a punctured key on $\{\tau^* \| m^* \| C^*\}$. We show that the input-output behaviour of the two programs are identical and the claim follows from the property of the indistinguishability obfuscator. More specifically, we show that the modified program will never abort in Step 5 for all the inputs and the PRF function indexed by K'_d in $\mathsf{Sim.Eval}_d^\times$ will not evaluate the point $(\tau^* \| m^* \| C^*)$. This is true because the C^* ends up with an addition gate $(+)$ and $\mathsf{Sim.Eval}_d^\times$ is the evaluation of a product gate $(\times)$. The string of the circuit will never equal to C^* in Step 5.

Suppose there exists a PPT adversary $\mathcal{A}$ such that $\Pr[X_1^d] - \Pr[X_2^d] \geq 1/\text{poly}(\lambda)$, we now construct a PPT algorithm $\mathcal{B}$ to break the indistinguishability obfuscator. Receiving (τ^*, m^*, C^*) from $\mathcal{A}$, $\mathcal{B}$ first generates f, $d+1$ PRF keys $K_0, ..., K_d \leftarrow \mathsf{F.Key}(1^\lambda)$ and then computes $K'_d \leftarrow \mathsf{F.Puncture}(K_d, \{\tau^* \| m^* \| C^*\})$. $\mathcal{B}$ creates the verification programs $\{\mathsf{Verify}_i\}_{0 \leq i \leq d-1}$ and $\mathsf{Sim.Verify}_d$, then generates the evaluation programs $\{\mathsf{Eval}_i^+, \mathsf{Eval}_i^\times, \mathsf{Eval}_i^=\}_{i \in [d-1]}$, $\mathsf{Eval}_d^=$, Eval_d^+, $\mathsf{Eval}_d^\times$ and $\mathsf{Sim.Eval}_d^\times$. $\mathcal{B}$ submits $\mathsf{Eval}_d^\times$ and $\mathsf{Sim.Eval}_d^\times$ to his $i\mathcal{O}$ challenger and receives $i\mathcal{O}(\mathsf{Eval}^\times)$. $\mathcal{B}$ sets $sk = K_0$, $vk = (\{i\mathcal{O}(\mathsf{Verify}_i)\}_{0 \leq i \leq d-1}, i\mathcal{O}(\mathsf{Sim.Verify}_d))$ and $vk = (\{i\mathcal{O}(\mathsf{Eval}_i^+), i\mathcal{O}(\mathsf{Eval}_i^\times), i\mathcal{O}(\mathsf{Eval}_i^=)\}_{i \in [d-1]}, i\mathcal{O}(\mathsf{Eval}_d^+)$, $i\mathcal{O}(\mathsf{Eval}_d^=), i\mathcal{O}(\mathsf{Eval}^\times))$. $\mathcal{B}$ sends vk, ek to $\mathcal{A}$ and answers $\mathcal{A}$'s queries with $sk = K_0$. $\mathcal{B}$ outputs 1 if $\mathcal{A}$ wins. Note that if $\mathsf{Eval}^\times = \mathsf{Eval}_d^\times$, then $\mathcal{A}$ is in $\mathbf{Hyb}_1$, if $\mathsf{Eval}^\times = \mathsf{Sim.Eval}_d^\times$, then $\mathcal{A}$ is in $\mathbf{Hyb}_2$. □

Lemma 8. *Assuming $i\mathcal{O}$ is an indistinguishability obfuscator, then* $\Pr[X_2^d] \leq \Pr[X_3^d] + negl(\lambda)$.

Proof. The only difference between $\mathbf{Hyb}_2$ and $\mathbf{Hyb}_3$ is the difference between the programs $\mathsf{Eval}_d^=$ and $\mathsf{Sim.Eval}_d^=$. We note that the input-output behaviour of these two programs are identical. The argument is similar to Lemma 7. □

Lemma 9. *Assuming $\Pi_{d-1,n}$ is selectively unforgeable for circuits in $\mathcal{C}_{d-1,n}$,* i.e. $\Pr[X_0^{d-1}] \leq negl(\lambda)$, *$i\mathcal{O}$ is an indistinguishability obfuscator, and f is injective, then* $\Pr[X_3^d] \leq \Pr[X_4^d] + negl(\lambda)$.

Proof. Note that we assume the C^* ends up with an addition gate (+). The difference between $\mathbf{Hyb}_3$ and $\mathbf{Hyb}_4$ is the difference between the programs Eval_d^+ and $\mathsf{Sim.Eval}_d^+$. We note that these two programs differs on inputs (c_1, c_2), where $c_1 = (\tau_1, m_1, \sigma_1, C_1), c_2 = (\tau_2, m_2, \sigma_2, C_2)$, satisfying that $\mathsf{Sim.Eval}_d^+$ runs to Step 5 on (c_1, c_2) and $\tau^* = \tau_1 = \tau_2$, $m^* = m_1 + m_2$, $C^* = C_1 + C_2$.

Let $C^* \triangleq C_1^* + C_2^*$. It must be $C_1 = C_1^*$ and $C_2 = C_2^*$. Because σ_1, σ_2 both pass the verification, it therefore should be the case that $f(\sigma_i) = f(\mathsf{F.Eval}(K_{d-1}, \tau^* \| m_i \| C_i^*))$ for $i = 1, 2$. Since f is injective, this is equivalent to $\sigma_i = \mathsf{F.Eval}(K_{d-1}, \tau^* \| m_i \| C_i^*)$ for $i = 1, 2$. Due to the determinism of the PRF function, we know that the number of differing inputs equals to the number of the pairs (m_1, m_2) such that $m_1 + m_2 = m^*$. Thus there are at most $2|\mathcal{M}|$ possible number of inputs that differ these two programs. (If C^* ends up with a product gate, then this upper bound is $2|\mathcal{M}|$)

Let $c_1 = (\tau^*, m_1, \sigma_1, C_1^*), c_2 = (\tau^*, m_2, \sigma_2, C_2^*)$ be a differing input on these two programs. We claim that this input satisfies:

$$(\tau^*, m_1) \neq (\tau_j, C_1^*(m_{j1}, ..., m_{jk})) \text{ or } (\tau^*, m_2) \neq (\tau_j, C_2^*(m_{j1}, ..., m_{jk})) \text{ for all } j, \tag{1}$$

where $(m_{j1}, ..., m_{jk})$ are the messages queried by $\mathcal{A}$ corresponding to τ_j. Suppose not, *i.e.* there exists some j such that $(\tau^*, m_1) = (\tau_j, C_1^*(m_{j1}, ..., m_{jk}))$ and $(\tau^*, m_2) = (\tau_j, C_2^*(m_{j1}, ..., m_{jk}))$. This implies

$$(\tau^*, m^*) = (\tau^*, m_1 + m_2) = (\tau_j, C_1^*(m_{j1}, ..., m_{jk}) + C_2^*(m_{j1}, ..., m_{jk})) = (\tau_j, C^*(m_{j1}, ..., m_{jk})).$$

This contradicts the strategy of $\mathcal{A}$ in the selective unforgeability experiment.

Assume that there exists a PPT algorithm $\mathcal{A}$ such that $\Pr[X_3^d] - \Pr[X_4^d] \geq 1/\mathrm{poly}(\lambda)$, we now construct a PPT algorithm $\mathcal{B} = (\mathcal{B}_1, \mathcal{B}_2)$ to break the security of $\Pi_{d-1,n}$ for circuits in $\mathcal{C}_{d-1,n}$.

After receiving (τ^*, m^*, C^*) from $\mathcal{A}$, $\mathcal{B}_1$ chooses $(\bar{m}, \bar{C}) \leftarrow \mathcal{M} \times \{C_1^*, C_2^*\}$ uniformly at random. Notice that $(\tau^*, \bar{m}, \bar{C})$ satisfies (1), *i.e.* $(\tau^*, \bar{m}) \neq (\tau_j, \bar{C}(m_{j1}, ..., m_{jk}))$ for all queries $(\tau_j, m_{j1}, ..., m_{jk})$, with probability at least $1/(2|\mathcal{M}|)$. $\mathcal{B}_1$ gives $(\tau^*, \bar{m}, \bar{C})$ to $\mathcal{B}_2$, then $\mathcal{B}_2$ submits it to the challenger in the selective unforgeability experiment on $\Pi_{d-1,s}$ and receives the evaluation key $ek' = \{(i\mathcal{O}(\mathsf{Eval}_i^+), i\mathcal{O}(\mathsf{Eval}_i^\times), i\mathcal{O}(\mathsf{Eval}_i^=))\}_{i \in [d-1]}$, and the verification key $vk' = \{i\mathcal{O}(\mathsf{Verify}_i)\}_{0 \leq i \leq d-1}$. $\mathcal{B}_2$ passes (ek', vk') to $\mathcal{B}_1$.

$\mathcal{B}_1$ is viewed as a distinguisher for $i\mathcal{O}(\mathsf{Eval}_d^+)$ and $i\mathcal{O}(\mathsf{Sim.Eval}_d^+)$. $\mathcal{B}_1$ generates the PRF key $K_d \leftarrow \mathsf{F.Key}(1^\lambda)$ and the punctured key $K_d' \leftarrow \mathsf{F.Puncture}(K_d, \{\tau^* \| m^* \| C^*\})$. $\mathcal{B}_1$ then creates verification and evaluation keys $i\mathcal{O}(\mathsf{Sim.Verify}_d)$, $i\mathcal{O}(\mathsf{Sim.Eval}_d^\times)$ and $i\mathcal{O}(\mathsf{Sim.Eval}_d^=)$ as in $\mathbf{Hyb}_4$. Note that $\mathcal{B}_1$ gets the obfuscation $i\mathcal{O}(\mathsf{Verify}_{d-1})$ from $\mathcal{B}_2$, then he can embed it to the evaluation programs. After that, $\mathcal{B}_1$ generates the programs Eval_d^+ and $\mathsf{Sim.Eval}_d^+$ and submits them to his $i\mathcal{O}$ challenger and receives $i\mathcal{O}(\mathsf{Eval}^+)$. $\mathcal{B}_1$ sets the verification key $vk = (vk', i\mathcal{O}(\mathsf{Sim.Verify}_d))$ and the evaluation key $ek = (ek', i\mathcal{O}(\mathsf{Sim.Eval}_d^\times), i\mathcal{O}(\mathsf{Sim.Eval}_d^=), i\mathcal{O}(\mathsf{Eval}^+))$ and sends them to $\mathcal{A}$. $\mathcal{B}_1$ answers $\mathcal{A}$'s signing queries using $\mathcal{B}_2$'s signing oracle on the $\Pi_{d-1,n}$ scheme. If $(\tau^*, \bar{m}) \neq (\tau_j, \bar{C}(m_{j1}, ..., m_{jk}))$ for all the queried messages $(m_{j1}, ..., m_{jk})$ corresponding to τ_j, then all the queries of $\mathcal{A}$ can be answered correctly. $\mathcal{B}_1$ outputs 1 if $\mathcal{A}$ wins.

Note that if $\mathsf{Eval}^+ = \mathsf{Eval}_d^+$ then $\mathcal{A}$ is in $\mathbf{Hyb}_3$, if $\mathsf{Eval}^+ = \mathsf{Sim.Eval}_d^+$, then $\mathcal{A}$ is in $\mathbf{Hyb}_4$. Since $\mathcal{A}$ can distinguish $\mathbf{Hyb}_3$ and $\mathbf{Hyb}_4$ with advantage $1/\mathrm{poly}(n)$, then $\mathcal{B}_1$ can distinguish $i\mathcal{O}(\mathsf{Eval}_d^+)$ and $i\mathcal{O}(\mathsf{Sim.Eval}_d^+)$ with advantage $\frac{1}{2|\mathcal{M}| \cdot \mathrm{poly}(\lambda)}$. Because there are at most $2|\mathcal{M}|$ inputs differ these two programs and by the statement of Lemma 2, there exists an extractor E outputs a differing input (c_1, c_2) with probability $1 - \mathrm{negl}(\lambda)$.

As discussed above, (c_1, c_2) must satisfy condition (1). Without loss of generality, we assume $c_1 = (\tau^*, m_1, \sigma_1, C_1^*)$ satisfying $(\tau^*, m_1) \neq (\tau_j, C_1^*(m_{j1}, ..., m_{jk}))$ for all the queries $(\tau_j, m_{j1}, ..., m_{jk})$. Since $(\bar{m}, \bar{C})$ is chosen uniformly at random from $\mathcal{M} \times \{C_1^*, C_2^*\}$, then $(\bar{m}, \bar{C}) = (m_1, C_1^*)$ with probability $1/(2|\mathcal{M}|)$. Finally $\mathcal{B}_2$ outputs σ_1 in the experiment on $\Pi_{d-1,s}$. Note that since (c_1, c_2) is a

differing input, then it holds that $i\mathcal{O}(\mathsf{Verify}_{d-1})(\tau^*, m_1, \sigma_1, C_1^*) = 1$ and $C_1^* \in \mathcal{C}_{d-1,n}$. Therefore σ_1 is a valid forgery, which implies $\mathcal{B} = (\mathcal{B}_1, \mathcal{B}_2)$ is a PPT adversary such that $\Pr[X_0^{d-1}] \geq \frac{1}{2|\mathcal{M}|} \cdot (1 - \mathrm{negl}(\lambda))$. We conclude that for any PPT adversary $\mathcal{A}$, we have $\Pr[X_3^d] - \Pr[X_4^d] \leq \mathrm{negl}(\lambda)$. □

Lemma 10. *Assuming* F *is a puncturable* PRF*, then* $\Pr[X_4^d] \leq \Pr[X_5^d] + \mathit{negl}(\lambda)$.

Proof. Suppose there exists a PPT adversary $\mathcal{A}$ such that $\Pr[X_4^d] - \Pr[X_5^d] \geq 1/\mathrm{poly}(\lambda)$, we construct a PPT algorithm $\mathcal{B}$ to break the pseudorandomness of the puncturable PRF. Receiving (τ^*, m^*, C^*) from $\mathcal{A}$, $\mathcal{B}$ submits the punctured set $\{\tau^* \| m^* \| C^*\}$ to his PRF challenger. $\mathcal{B}$ gets K' and r from the challenger, where $K' \leftarrow \mathsf{F.Punture}(K, \{\tau^* \| m^* \| C^*\})$ and either $r = \mathsf{F.Eval}(K, \tau^* \| m^* \| C^*)$ or r is uniformly random from $\{0,1\}^\lambda$. Then $\mathcal{B}$ generates the signing key $sk = K_0$, the verification key vk and the evaluation key ek as in $\mathbf{Hyb}_4$ by setting $K'_d = K'$ and $z^* = f(r)$. $\mathcal{B}$ sends (vk, ek) to $\mathcal{A}$ and answers $\mathcal{A}$'s query using sk. Finally, $\mathcal{B}$ outputs 1 if $\mathcal{A}$ wins. Note that if $r = \mathsf{F.Eval}(K, \tau^* \| m^* \| C^*)$, then $\mathcal{A}$ is in $\mathbf{Hyb}_4$, if $r \leftarrow \{0,1\}^\ell$, then $\mathcal{A}$ is in $\mathbf{Hyb}_5$. □

Lemma 11. *Assuming* f *is a one-way function, then* $\Pr[X_5^d] \leq \mathit{negl}(\lambda)$.

Proof. Let $\mathcal{A}$ be a PPT adversary that outputs a valid forgery in $\mathbf{Hyb}_5$, we construct a PPT algorithm $\mathcal{B}$ to break the one-wayness of f. Receiving $z^* = f(r^*)$ with $r^* \leftarrow \{0,1\}^\ell$ from the one-way function challenger, $\mathcal{B}$ generates the signing key sk, verification key vk and evaluation key ek by using f and z^* as in $\mathbf{Hyb}_5$. $\mathcal{B}$ answers the signing queries with sk and outputs whatever $\mathcal{A}$ outputs. Note that if $\mathcal{A}$ outputs a valid forgery σ^*, it must satisfy $\mathsf{Sim.Verify}_d(\tau^*, m^*, \sigma^*, C^*) = 1$. Therefore, we have $f(\sigma^*) = z^*$, and σ^* is a pre-image of z^*. □

Theorem 2 follows from Lemma 6, 7, 8, 9, 10, 11. □