MPC-Friendly Commitments for Publicly Verifiable Covert Security

Nitin Agrawal
nitin.agrawal@cs.ox.ac.uk
University of Oxford

James Bell
jbell@turing.ac.uk
The Alan Turing Institute

Adrià Gascón
adriag@google.com
Google

Matt J. Kusner
m.kusner@ucl.ac.uk
University College London

ABSTRACT

We address the problem of efficiently verifying a commitment in a two-party computation. This addresses the scenario where a party P1 commits to a value x to be used in a subsequent secure computation with another party P2 that wants to receive assurance that P1 did not cheat, i.e. that x was indeed the value inputted into the secure computation. Our constructions operate in the publicly verifiable covert (PVC) security model, which is a relaxation of the malicious model of MPC appropriate in settings where P1 faces a reputational harm if caught cheating.

We introduce the notion of PVC commitment scheme and indexed hash functions to build commitments schemes tailored to the PVC framework, and propose constructions for both arithmetic and Boolean circuits that result in very efficient circuits. From a practical standpoint, our constructions for Boolean circuits are 60× faster to evaluate securely, and use 36× less communication than baseline methods based on hashing. Moreover, we show that our constructions are tight in terms of required non-linear operations, by proving lower bounds on the nonlinear gate count of commitment verification circuits. Finally, we present a technique to amplify the security properties of our constructions that allows to efficiently recover malicious guarantees with statistical security.

KEYWORDS

Privacy-preserving deep learning; Committed MPC

1 INTRODUCTION

Secure multi-party computation (MPC) methods have undergone impressive improvements in the last decade. Advances in the scalability of garbled circuit protocols [8, 40, 41], commitment schemes [15], and oblivious transfer [6, 30] have transformed the range of applications for MPC [26]. In particular, a significant amount of research efforts have been recently devoted to finding efficient MPC protocols for training and evaluation of widely-deployed machine learning (ML) models [1, 16, 18, 28, 29, 31, 36, 38]. These works enable collaborative training, as well as private predictions, where users can get predictions from a confidential model while preserving the privacy of their input.

Generally speaking, the guarantee of an MPC computation is that the inputs of the participants remain private to other parties, but that does not prevent parties to choose their inputs in an arbitrary way, e.g., in the well-known millionaires problem, nothing prevents a millionaire from lying about their wealth. Going back to the private prediction application mentioned above: nothing prevents the model owner from modifying the model arbitrarily. This is a problem in settings where the model has to satisfy certain non-functional constraints such as safety, fairness, or privacy. These constraints undermine accuracy (as often measured in ML) and thus the model owner may have an incentive to switch the model. This exact problem and, more generally, model certification, was tackled recently by Kilbertus et al. [24] and Segal et al. [37]. Both these works show that commitments verified in MPC can help here. For example, consider a service provider offering dietary or exercise recommendations based on personal data. Users may require the service provider to commit to a recommendation algorithm that is certified not to make harmful recommendations (which could have been verified and signed by a regulator). More formally a user requires the following 2-party functionality: the service provider (P1) commits to an input x by producing commitment c and sends c to the user (P2). Later, P2 uses c to verify that x is being used by P1 inside an MPC protocol between both parties. We call this *MPC on committed data*. We describe an application of this framework to certified predictions in Appendix D, along with an empirical example on realistic data showing that heuristic methods that do not ensure that the model does not change will fail. Concretely, we show that changing a single parameter in a fair model results in an unfair model with increased accuracy.

So far current work on this uses standard collision-resistant hash functions such as SHA-256 [24] and SHA-3 [37] to produce and then verify commitments in MPC. However, these methods do not take advantage of two key properties of this setting: 1. *Interactivity*: given that an MPC protocol needs to be run between the user and service provider to compute some functionality (e.g., a recommendation), it is possible to leverage the interactivity of the protocol to construct a commitment; 2. *Reputation of service provider*: as this computation involves a service provider who relies on users for profit, a protocol can be constructed so that cheating would harm the reputation of the service provider, using ideas from Publicly-Verifiable Covert (PVC) security [5].

Based on these properties we make a simple observation: one can detect if an input x to a Boolean MPC protocol has been changed with probability $1/2 - \epsilon$ (for arbitrarily small $\epsilon > 0$) using a simple additional Boolean circuit as part of the protocol (more details on such circuits are in Figure 2). The idea is that in MPC a hash can be efficiently constructed by using inputs from both parties, an idea we call *indexed hash functions*. These functions allow one to build MPC commitments in the PVC setting that analytically and

experimentally outperform prior approaches by as much as 60× in runtime and 36× in communication.

Other Related Work. **?**] also discusses the problem of input validity, using efficient SFE protocols. In particular, the solution utilizes universal hash functions and committed OT. The protocol specifically improves the performance of garbled circuit based secure function evaluation for cases where sub-circuits depend on only one party's input. Concurrent with [**?**], **?**] propose a solution performing predicate checks followed by secure evaluation of an arbitrary function, if the inputs pass the verification. However, both these works focus on malicious security, different from our proposal in PVC security model, utilizing properties 1,2. In particular, our approach does not fit their paradigm because our commitment verification has a private input. However, our results for maliciously secure setting (section 8) are compatible with their optimizations.

In this paper we give a technical overview of our paper, and describe our contributions. We then introduce indexed hash functions and give efficient constructions for them. We describe how to use these hash functions to enable MPC on committed data. We give an analytic and experimental comparison with prior work. We derive lower bounds on the number of AND gates necessary for indexed hash functions, demonstrating that some of our constructions are as efficient as possible. A natural question is if the security guarantees of indexed hash functions can be extended to computational security. We answer this question affirmatively: we give a construction and present complexity results. Finally we describe initial ideas of extensions of this approach for arithmetic circuits.

Although we chose to motivate our contribution from the perspective of certified predictions, our results are general, and essentially provide constructions of commitment schemes tailored for PVC security, along with efficient implementations in MPC.

2 PRELIMINARIES

We give a brief background on key ideas we will use in the paper.

Hash functions and pseudo-randomness. We consider a hash function to be a function $h : \{0,1\}^* \to O$ for some finite output space O. Informally, h is *collision resistant* if no adversary is capable of finding two distinct inputs with the same image except with negligible probability. The formal definition requires talking about families of hash functions [22]. A pseudo-random number generator, or PRNG, is a function $\mathsf{prng} : K \times \mathbb{N} \to \{0,1\}^*$ which maps (k, b) to a bit-string of length b. Both hash functions and PRNGs can be modelled as random oracles i.e. as a uniformly random mapping from their inputs to their outputs.

Publicly-verifiable covert (PVC) security. Covert security [7] weakens the malicious security setting by guaranteeing that a cheating party (who may behave arbitrarily) will be caught by the other party with a probability, p, referred to as the covert security parameter. The motivation for covert security is that if certain parties have a reputation to preserve, then the risk associated with being caught, outweighs the benefit of cheating. This allows faster protocols than malicious security [7, 12, 19, 25]. PVC security was introduced by Asharov and Orlandi [5]. It, with probability p, provides a publicly-verifiable proof of cheating, which allows greater reputational harm and possibly legal repercussions for a cheater.

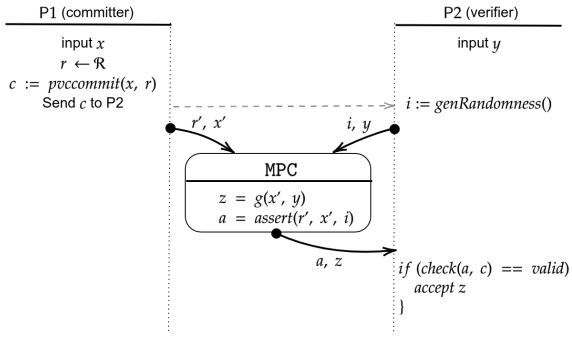

Figure 1: The diagram shows a multi-party computation with a committed input x, as enabled by our constructions. Party 1 (the committer) holds an input x, for which it generates a commitment c and sends it to party 2 (the verifier). The commitment is randomized using r to ensure privacy for x, i.e., that the commitment is hiding. Later on, the parties engage in a secure computation of a generic function g, where party 1 inputs x', and party 2 inputs input y. For the purpose of verifying that $x = x'$, party 2 derives a challenge i from c, and the MPC returns a certificate a that can be checked by P2. This guarantees to party 2 that g is evaluated on the value x to which party 1 had previously committed.

3 TECHNICAL OVERVIEW

Hashing is a useful primitive to implement in MPC, as it enables privacy-preserving consistency checks, and thus *MPC on committed data*. This paper focuses on efficient implementation of this functionality, depicted abstractly in Figure 1. The high-level goal is to enable a party P1 to commit to a *private* value x at some point in time and later, when engaging in a secure computation of a function g with a second party P2, provide the assurance to P2 that P1 inputs the committed value x into the computation and not some other value. This is modelled, analogously to commitment schemes, by three algorithms: pvccommit, assert, and check. As shown in Figure 1, pvccommit outputs a commitment to an input, later in the MPC, assert is run. As we will see next, naively, assert could simply compute pvccommit, but our constructions will leverage randomness in assert. Otherwise the algorithms correspond to a standard commitment scheme, in that they should satisfy the standard *binding and hiding* properties. The function check is used to interpret the results and see whether cheating has occurred. The verification is split between assert and check and a single commitment allows for arbitrarily many secure computations, in the future.

The functionality of Figure 1 has appeared in previous works on *certified predictions* [24, 37] where x is a confidential machine learning model owned by P1 that has been checked by a certifying authority to have certain properties, e.g., fairness. In those applications, g corresponds to model evaluation and users receive predictions using the certified model x. MPC on committed data directly enables this functionality, with the model owner and the users playing the roles of parties P1 and P2 in Figure 1, respectively. We describe and motivate this application in Appendix D.

Baseline protocol. To see how a hash function h, e.g., SHA3 in practice, can be used to implement MPC on committed data, consider the following instantiation of pvccommit, assert, and check.

P1 can just choose a random r and have pvccommit(x, r) and assert(x, r) be $h(x||r)$. Then, check just checks that they are equal. Both pvccommit and check are efficient in this instantiation, so one would want h to have an efficient MPC protocol.

MPC-friendly hashing. The works on fairness certification of Segal et al. [37] and Kilbertus et al. [24] propose the above baseline construction. Segal et. al concretely instantiate h with the Keccak-F function, which is the basis of the SHA3 standard. That function takes a 1600 bit input and can be implemented by a Boolean circuit of 38400 AND gates i.e. 24 AND gates per input bit [37**?**]. Using a Merkle tree for succinctness of c, the total number of hashes is twice the number of input blocks, resulting in 48 AND gates per input bit of x. Thus assert would result in an overhead of 48 AND gates/bit in this instantiation. Alternatively, using SHA3-256 in sponge mode results in an overhead of roughly 35 AND gates/bit. On the other hand, using an MPC-optimized hash (but new and susceptible [14]) LowMCHash-256 [3] in sponge mode roughly take up 14 AND gates/bit. Note that AND gate counts, and non-linear gate counts in general, are a standard reference for computation time in MPC, and secure computation in general. In this work we propose efficient MPC-friendly commitments schemes based on hashing, with a focus on Publicly Verifiable Covert (PVC) security.

Our starting observation is that executing a collision-resistant hash function such as SHA-256 in a PVC-secure protocol is an overkill: Note that ensuring that commitments are binding, i.e., that P1 in Figure 1 can not generate x', r' such that $x \neq x'$ and the verification passes, up to negligible probability, but then ensuring that the subsequent MPC is secure only up to a constant probability p leaves some potential room for weakening the binding guarantee of the commitment scheme to favor efficiency. To take advantage of the PVC setting, we observe that assert must in some way receive randomness from P2, as it is against this randomness that P1 will have probability p of being caught. We design assert function that leverage this observation, and result in an overhead of the assert circuit as low as *half an AND gate per bit of the input* x, even when p is close to 1, including $1 - 2^{-\sigma}$ for a statistical security parameter σ.

3.1 Contributions

We introduce the notion of an *indexed hash function*. This, roughly speaking, is a function that produces a hash of an input x, given a random value r and an index i from a domain $\mathcal{I}$. The index of the hash function plays the role of the randomness chosen by P2 mentioned above. We build indexed hash functions from any collision-resistant hash function h and prove properties related to collision resistance that allow us to construct PVC commitments from indexed hash functions, and use them for achieving the functionality in Figure 1 with PVC security.

Given an indexed hash function H, our proposed PVC commitment schemes follow the following high-level structure: pvccommit computes $(H(j, r, x))_{j \in I}$, i.e. the hash evaluated at all indices, the verifier selects an index $i \in \mathcal{I}$, assert computes $H(i, r, x)$, and check checks that this value is correct. This check fails (in the sense of giving a false positive) with a probability upper bounded by $1 - p$ (p is called the covert security parameter). This fact is formalized by reducing an appropriate notion of collision resistance of H to the collision resistance of h.

The efficiency of our scheme relies on the fact that our constructions for indexed hash functions are very efficient to be evaluated in MPC, requiring a very small number of XOR and AND gates with respect to the input size and thus inducing a very small overhead when evaluated in MPC. Next, we summarize the organization of the paper, highlighting the key contributions of each section.

A construction for Boolean circuits (Section 4). We give a construction for Boolean circuits that can achieve a covert security parameter arbitrarily close to 1/2, which asymptotically requires only half an AND gate per bit. Furthermore, in practice, it requires less than one AND gate per bit for moderately sized inputs.

PVC (Section 5). We define PVC commitment schemes and their security properties, propose a secure instantiation based on indexed hash functions, and show how to use them for committed PVC MPC.

Experimental Evaluation. (Section 6) We fully implement our most practical construction, and compare it with the baseline approach (both instantiated with SHA3 and an MPC-friendly hash function LowMCHash). Our experiments show a 60× speed up and 36× less communication in the resulting assert compared to SHA3. It takes $\sim$ 1 MB and $<$ 15 minutes to commit million bit inputs.

Lower bounds (Section 7). We show the optimality of our construction by proving lower bounds for both our approach and the baseline hash based approach. In particular, we show that if the resulting hash is required to be even remotely succinct (to have size $< 99x/100$) then (i) the indexed hash function must require at least half an AND gate per bit asymptotically, and (ii) with an ordinary hash function at least one AND gate per bit is required. We also show that even if we remove the assumption on the size of the hash at least 1/5 of an AND gate per bit would be required.

A way to amplify security (Section 8). We give a way to achieve covert security parameter $1 - 2^{-\sigma}$, and thus full statistical security, in both of the Boolean and arithmetic cases. While the method relies on repeating our constructions to amplify their probabilistic guarantees, we are able to maintain the asymptotic half a non-linear gate per bit. This result is mostly of theoretical relevance at the moment, as it does not beat the state of the art of MPC-friendly hash functions for input sizes that are currently practical.

A construction for Arithmetic circuits (Section 9). We give an analog to our Boolean construction for arithmetic circuits which can achieve a covert security parameter arbitrarily close to 1 with only half a MULT gate per input element asymptotically. This is also a practical improvement over the best known ordinary hash functions [3]. However the security parameter can not be taken to be 1 minus negligible.

4 INDEXED HASH FUNCTIONS

In this section we will introduce the primitive we will use to build our commitments: *indexed hash functions*. We do so informally and then formally, and give some examples of indexed hash functions. The examples will show that secure indexed hash functions have much smaller circuits than ordinary secure hash functions.

Like an ordinary secure hash function, an indexed hash function takes an input from some space $\mathcal{X}$ and produces an output in a space O. When working with a specific $x \in \mathcal{X}$ we will denote by n the bitlength of x.

The whole idea is to take advantage of the fact that the verifier (P2) can have an input to the indexed hash. We call this input the *index* of the hash function and denote it by i, drawn from an index set $\mathcal{I}$. If the wrong input from $\mathcal{X}$ is used during verification, then at least a fixed fraction of indices i will result in an incorrect hash.

We need to ensure that the hash is hiding, otherwise if $x_1, x_2 \in \mathcal{X}$ were the possible inputs by the committer, then by computing the hash of x_1 and x_2 the verifier could learn which was the true input from the output. Thus the committer must provide an extra random input r from some set $\mathcal{R}$ containing enough entropy to hide the true input. Hence we define indexed hash functions as functions taking an index i, a random nonce r, and an input x.[1]

Definition 4.1. An *indexed hash function* is a function $H : \mathcal{I} \times \mathcal{R} \times \mathcal{X} \to O$.

4.1 Collision resistance

Next, we formally define two properties for indexed hash function that we require: (i) the hiding property, i.e., not leaking information about x, and (ii) a notion of collision resistance which we call *q-collision boundedness*. For (i) we can use the definition from classical commitment schemes: we say H is *hiding* if an adversary learns a negligible amount about an input x from learning the value of $H(i, r, x)$ for all $i \in \mathcal{I}$, and a uniformly random (and secret from the adversary) $r \in \mathcal{R}$.

For (ii), we start by defining the concept of a q-collision, which denotes a pair of inputs on which H collides for at least a fraction q of all possible indices $|\mathcal{I}|$.

Definition 4.2. Let $q \in [0, 1]$. We call a quadruple r, x, r', x' a *q-collision* of H if $x \neq x'$ and

$$|\{i \in \mathcal{I} \,|\, H(i, r, x) = H(i, r', x')\}| \geq q|\mathcal{I}|$$

We can now define our notion of collision resistance. Informally, H is *q-collision resistant*, if adversaries are unable to find a q-collision of H except with negligible probability. We formalize this using families of indexed hash functions, in turn indexed by a key $k \in K$ generated by a generator G taking a computational security parameter λ. This is similar to the standard definition of a family of collision-resistant hash functions. Moreover, we say that H is *q-collision bounded* if it is q'-collision resistant for every $q' > q$.

The security parameter λ. We use a single computational security parameter λ for all aspects of our constructions. This includes their underlying collision resistant hash function, as well as the size of the source of randomness $\mathcal{R}$ and the set of indices $\mathcal{I}$. In particular, $|\mathcal{I}|$ is polynomial in λ and $|\mathcal{R}|$ is exponential in λ in all constructions. Thus our security is formalized in terms of polynomial time adversaries w.r.t. λ, and whose advantage should be bounded by a negligible function in λ. Note that this implies that an attacker is allowed to iterate over $\mathcal{I}$, and in fact in practice we will ensure that $\mathcal{I}$ is as small as possible for efficiency.

[1] We call these hash functions because they have output smaller than their input and, even with the insertion of randomness that is not technically part of the function, a single evaluation of this function would not create a commitment.

Definition 4.3. Given a generator G, security parameter λ, and key $k = G(\lambda)$, a family $\{H_k\}_{k \in K}$ is q-collision resistant if, for any probabilistic polynomial time algorithm A we have that

$$\mathbb{P}[A(k) \text{ is a } q\text{-collision of } H_k] < \text{negl}(\lambda)$$

Given this, we can define our main notion of collision as follows.

Definition 4.4. A family $\{H_k\}_{k \in K}$ is q-collision bounded if it is q'-collision resistant for all $q' > q$.

Any family $\{H_k\}_{k \in K}$ that is q-collision resistant is also *q-collision bounded*. This is simply due to the definition of q-collisions: any q-collision is also a q'-collision for all $1 \geq q' > q$.

This property will be useful later because by choosing an index uniformly at random we can distinguish between any two inputs $x, x' \in \mathcal{X}$, s.t. with probability at least $1 - q$ by looking at a hash. We can now make the hiding property precise. That H is hiding will be proved under the assumption that h is a random oracle.

Definition 4.5. A family of indexed hash functions $\{H_k\}_{k \in K}$ is hiding if for any polynomial time algorithm A and any $x, x' \in \mathcal{X}$, for a uniformly random choice of $r \in \mathcal{R}$ we have

$$\mathbb{P}(A(k, (H_k(i, r, x))_{i \in \mathcal{I}}) = 1) = \mathbb{P}(A(k, (H_k(i, r, x'))_{i \in \mathcal{I}}) = 1) + \text{negl}(\lambda)$$

As mentioned above, we will construct our indexed hash functions by building them from an ordinary secure hash function h. We do so because it will allow us to prove q-collision resistance of H via a (ptime) reduction to collision resistance of h. To argue about collision boundedness we formalize the notion of a construction.

Definition 4.6. A *construction* of an indexed hash function is a function C which given a hash h and the security parameter λ, returns an indexed hash H.

We say that C, *preserves q-collision boundedness* if there is an (efficient) algorithm which, given a q-collision of C, returns a collision of h. Thus if a powerful adversary is unable to find a collision in some fixed hash function h, then it is reasonable to assume they cannot find a $q'|\mathcal{I}|$-sized collision in $\mathcal{H}$ for any $q' > q$.

Definition 4.7. We say that C *preserves q-collision boundedness* if $\{h_k\}_K$ being collision resistant implies that $\{H_k = C(h_k)\}_K$ is q-collision bounded.

4.2 Constructions

We will now give four constructions of indexed hash functions denoted C_0 through C_3. Our construction 3 is the most practical and the one we would recommend to use (we will use this construction in our experiments in Section 6), but we include all four to build up ideas incrementally. Our goal is to derive indexed hash functions that 1. *Are efficient to implement in MPC*: this aspect of the constructions in this section is captured in terms of *the number of AND and XOR gates* of their corresponding Boolean circuit implementations (in Section 9 we give a construction for arithmetic circuits); 2. *Have a small index domain $\mathcal{I}$*: this directly corresponds to the commitment size of the PVC commitment schemes that we will build on top of them. For these constructions the value of q (i.e., for which constructions are q-collision bounded) is one minus the security parameter of the PVC commitment scheme derived later.

Construction	C_0 (baseline)	C_1	C_2	C_3 (main)
# ANDs	$C_N n$	$n + C_N \frac{n}{b}$	$\frac{n}{2} + C_N \frac{n}{b}$	$\frac{n}{2} + C_N \frac{n}{b}$
# XORs	$C_L n$	$n + C_L \frac{n}{b}$	$\frac{3n}{2} + C_L \frac{n}{b}$	$\frac{3n}{2} + C_L \frac{n}{b}$
$\lvert\mathcal{I}\rvert$	1	2^b	2^b	$\frac{b+\lambda+1}{2\epsilon^2}$
q	1	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2} + \epsilon$

Table 1: Let H be an indexed hash function resulting from a construction, using h as the underlying collision resistant hash function. This table shows (i) the size of a Boolean circuit for H in terms of number of bits n of the input (omitting lower order $o(n)$ additive terms), where C_N and C_L denote the number of AND and XOR gates of h, respectively; (ii) the size of $\mathcal{I}$, and (iii) the value of q for which q-collision resistance holds. The parameter b can be chosen to be any even positive integer and ϵ can be taken to be any positive value.

We summarize computation, size, and allowed q values in Table 1. In this section we have $q = 1/2$ and $q = 1/2 + \epsilon$ (for arbitrarily small $\epsilon > 0$), but in Section 8 we show how to achieve arbitrarily small q.

Blueprint for our constructions. Consider an indexed hash function H taking an index i, randomness r, and input x. All our constructions are parametrized by a *block size* $b \in [|x|]$ and a *block digest function* d. The latter takes (i) a binary encoding of i and (ii) a bitstring of size b, and outputs a *single* bit, i.e. $d : \mathcal{I} \times \{0,1\}^b \mapsto \{0,1\}$. The indexed hash function is defined to be the result of

(1) splitting x into n/b consecutive blocks of b bits (if $|x|$ is not a multiple of b, it can be padded with zeros),
(2) applying $d(i, \cdot)$ to each block x_j, and
(3) outputting the length n/b bitstring resulting from concatenating all digested bits $d(i, x_j)$.

We denote the result of processing an input i, x as in steps 1-3 by $\texttt{process}_{b,d}(i, x)$. Digest function d determines the size of the index set $\mathcal{I}$.

Then, each construction C is defined by a block size b, digest function d, an ordinary collision resistant hash function h, and a set of random masks $\mathcal{R}$ (which we always take to be $\{0,1\}^\lambda$) as:

$$C(h, \lambda)(i, r, x) = h(r||i||\texttt{process}_{b,d}(i, x)) \tag{1}$$

When presenting the three constructions in this section we will denote the digest function associated with C_j by d_j. The motivation behind this presentation is simplicity, as it is now enough to define d_1, d_2, d_3, and the arguments in our proofs only need to refer to d_j.

The hiding property. We can prove the hiding property (Def. 4.5) without knowing anything about d and thus the size of $\mathcal{I}$, so we do this in generality for all the constructions.

Theorem 4.8. *Suppose C is given by Equation 1. If $\{h_k\}_{k\in K}$ is a family of random oracles then $\{C(h_k, \lambda)\}_{k\in K}$ is hiding.*

See Appendix A for the proof of this theorem.

Collision boundedness. For each C_j we propose, we will show q-collision boundedness of $H = C_j(h, \lambda)$. This will be done by showing that for at most $q|\mathcal{I}|$ indices i we have $\texttt{process}_{b,d}(i, x) = \texttt{process}_{b,d}(i, x')$. That this is a sufficient condition for q-collision boundedness is shown in the following theorem.

Theorem 4.9. *Let $q \in [0, 1]$. Suppose that for any $x \neq x'$ and for any sufficiently large λ,*

$$|\{i \in \mathcal{I} \,|\, \texttt{process}_{b,d}(i, x) = \texttt{process}_{b,d}(i, x')\}| \leq q|\mathcal{I}|$$

then the construction in Equation 1 preserves q-collision boundedness.

Proof. Let $\{h_k\}_{k\in K}$ be a collision resistant family of hash functions, and $H_k = C(h_k, \lambda)$. Suppose that the hypothesis of the statement holds but there exists a polynomial time algorithm A which finds a q-collision in H_k with non-negligible probability. We show next that $\{h_k\}_{k\in K}$ is not collision resistant: a contradiction.

Let $k = G(\lambda)$. For sufficiently large λ, a probabilistic ptime (in λ) algorithm B for finding a collision in h with non-negligible probability is given by the following. Given k, B computes $(r, x, r', x') = A(k)$. If (r, x, r', x') is a q'-collision for some $q' > q$ (this happens with non-negligible probability) then B computes an index i such that $\texttt{process}_{b,d}(i, x) \neq \texttt{process}_{b,d}(i, x')$ but $H_k(i, r, x) = H_k(i, r', x')$. Note that such an i must exist with probability 1 and can be found by exhaustion in time $O(\mathcal{I})$ (and thus $O(\lambda)$). By Equation 1 we now have that $r||i||\texttt{process}_{b,d}(i, x)$ and $r'||i||\texttt{process}_{b,d}(i, x')$ form a collision in h_k. The algorithm B outputs this collision. The fact that B succeeds with non-negligible probability contradicts the collision resistance of the family $\{h_k\}_{k\in K}$. □

We now proceed to present each construction C_j by specifying the length of $\mathcal{I}$ and the digest function d_j to sub into Equation 1. Recall that all constructions are summarized in Table 1.

Construction 0. Firstly let us consider a trivial construction. Let $\mathcal{I}$ be the set containing only the empty string ϵ and let $d_0(i, x) = x$ and the block size be $b_0 = |x|$. Let

$$h(r||\epsilon||\texttt{process}_{b_0,d_0}(i, x)) = h(r||x).$$

Theorem 4.10. *C_0 preserves 0-collision boundedness.*

Proof. By Theorem 4.9 this is immediate as the identity function has no collisions, and thus the condition of that theorem holds for $q = 0$. □

This construction is very simple and d_0 is trivial to compute. However, we can improve over this by making the digest function d compress the input so that h only has to be computed on an input much smaller than x, resulting in a more efficient circuit.

Construction 1: n AND gates. The digest function for construction 1 is shown in Figure 2 (left). As above, this construction is parameterised by a block size b. Let $\mathcal{I} = \{0,1\}^b$ and let x_j be the block containing bits jb through $jb + b - 1$, inclusive, of x (padding x with zeros to length a multiple of b). We use & to denote bitwise AND, and let parity map bit strings to the XOR of all their bits. Then we define $d_1(i, x_j) = \texttt{parity}(x_j \& i)$.

Theorem 4.11. *C_1 preserves 1/2-collision boundedness.*

Proof. If x and x' differ, they differ in some block j. Let $\texttt{process}_{b,d}(i, x)_j$ be the jth bit of $\texttt{process}_{b,d_1}(i, x)$. Note that $\texttt{process}_{b,d_1}(i, x)_j = \texttt{process}_{b,d_1}(i, x)_j$ if and only if $d_1(i, x_j) = d_1(i, x_j)$, which happens if and only if $\texttt{parity}((x_j \oplus x'_j)\&i) = 0$ (as conjunction distributes over exclusive or). As $x_j \oplus x'_j \neq 0$, we have that $\texttt{process}_{b,d_1}(i, x) = \texttt{process}_{b,d_1}(i, x)$ holds for exactly half of the possible values of i. The result is then immediate from Theorem 4.9. □

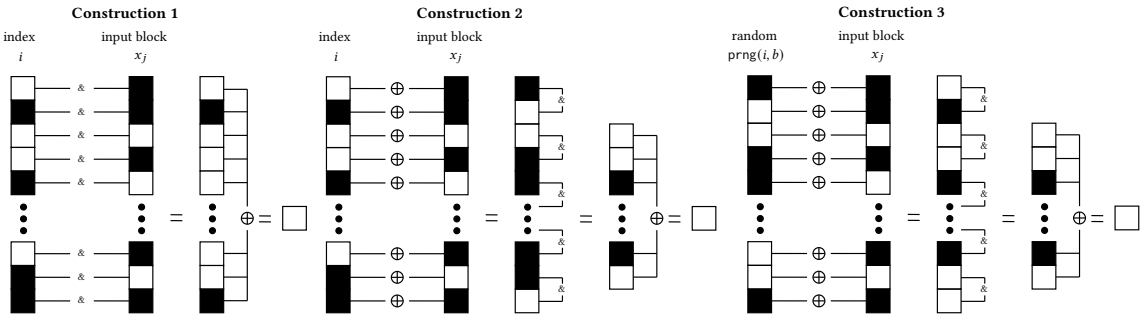

Figure 2: A depiction of digest functions d_1 (left), d_2 (center), and d_3 (right).

Note that d_1 (and thus $\mathtt{process}_{b,d_1}(i,x)$) has only one AND gate per bit of input and the output of $\mathtt{process}_{b,d_1}(i,x)$ has size $\lceil n/b \rceil$ (see Figure 2). Thus for large n and b this construction can asymptotically be computed with a number of AND gates arbitrarily close to n (and n XOR gates). However, as we will see in the lower bounds section (Section 7) only $n/2$ AND gates are needed for any collision resistant indexed hash function. This motivates looking for the following construction which closes this gap.

Construction 2: $\frac{n}{2}$ AND gates. We wish to avoid using an AND gate for each bit of the input but still need some nonlinearity in d. So the idea is to combine two bits of the input together using a single AND gate. If the raw input bits went directly into the AND gate then the adversary would sometimes be able to change them in a way that definitely would not change the output of the gate. Instead we will first XOR each bit with a bit from i. Now the adversary cannot tell whether changing a certain bit will change the output.

Let $\oplus$ denote bitwise XOR. Let y be a bitstring of length $2m$ and let y_j be the jth bit of y, we define $\mathsf{andreduce}(y)$, as the concatenation of $y_{2j} \& y_{2j+1}$ for all $j \in (0, ..., m-1)$. Then we define $d_2(i, x_j)$ to be $\mathsf{parity}(\mathsf{andreduce}(x_j \oplus i))$, see Figure 2 (center).

A *bent function* has the property that for a fixed linear change to its input, the output of the function would change for exactly half of all starting inputs [35]. The following theorem boils down to showing that parity $\circ$ andreduce is a bent function.

Theorem 4.12. *C_2 preserves $1/2$-collision boundedness.*

Proof. By Theorem 4.9 it suffices to show that if $x_j \neq x'_j$, i.e. x and x' differ in the jth block, then $e_j := d_2(x_j) \oplus d_2(x'_j) = \mathsf{parity}(\mathsf{andreduce}(x_j \oplus i)) \oplus \mathsf{parity}(\mathsf{andreduce}(x'_j \oplus i))$ is a uniformly random bit for a randomly chosen $i \in \mathcal{I}$.

Given that $x_j \neq x'_j$ we assume WLOG that they differ in at least one of the first two bits. The first bit of $\mathsf{andreduce}(x_j \oplus i)$ is 1 if and only if the first two bits of i are the bitwise not of the first two bits of x_j, therefore it is 1 with probability $1/4$. Similarly the first bit of $\mathsf{andreduce}(x'_j \oplus i)$ is 1 with probability $1/4$ and they can not both be 1 at once. Therefore they differ with probability $1/2$. Further they are independent of all but the first two bits of i and thus of the rest of $\mathsf{andreduce}(x_j \oplus i)$ and $\mathsf{andreduce}(x'_j \oplus i)$. It follows that e_j is uniformly random. □

This construction can approach arbitrarily close to half an AND gate per bit of input by choosing b appropriately (see Table 1). However, suppose h is SHA3-256, which requires ≈ 35 AND gates per bit of input (and thus $C_N \approx 35$ in Table 1). Then for H to have less than one AND gate per bit we would have to take $b \geq 70$. This would give $|\mathcal{I}| \geq 2^{70}$. Recall that $|\mathcal{I}|$ directly corresponds to the size of our commitments, and thus $|\mathcal{I}| = 2^b$ is impractical in computation and communication/storage. Construction 3 sacrifices a small amount of collision resistance in order to reduce the size of $\mathcal{I}$.

Construction 3: main result. The digest function d_3 is shown in Figure 2 (right), and is analogous to d_2, but instead of using $i \in \mathcal{I} = \{0,1\}^b$ directly, we use a pseudo-random number generator prng to expand i into length b strings to xor with the blocks of x. This corresponds to re-interpreting the set of indices $\mathcal{I}$ as the set of seeds of prng. This replaces the need for $|\mathcal{I}| = 2^b$ from construction 2 by a much smaller $\mathcal{I}$ of size linear in b and λ.

Concretely, let $\mathsf{prng} : \mathcal{Q} \to \{0,1\}^b$ be a pseudo-random number generator, with an arbitrarily large keyspace $\mathcal{Q}$. Let an evaluation of prng on key i as $\mathsf{prng}(i, b)$ denote that the prng stretches the input to length b. For practical purposes we can think of $\mathcal{Q} = \{0,1\}^{128}$ if, for example, we instantiate prng with AES (in counter mode) with 128-bit keys. Let $\mathcal{I}$ be a subset of $\mathcal{Q}$. For construction 3 we define $d_3(i, x)$ to be $\mathsf{parity}(\mathsf{andreduce}(x_i \oplus \mathsf{prng}(i, b)))$.

Because we do not use a uniformly distributed mask on each block we do not get $1/2$-collision boundedness. However, we can get arbitrarily close to that by increasing the size of $\mathcal{I}$. In particular $|\mathcal{I}|$ need only grow linearly in b, as shown in the following theorem.

Theorem 4.13. *If* prng *is a random oracle, then given any $q > 1/2$, there exists a choice of $|\mathcal{I}|$ such that with probability $1 - 2^{\sigma}$ (over the randomness of* prng*), construction* 3 *preserves q-collision boundedness. Specifically, it suffices to take*

$$|\mathcal{I}| \geq \frac{1}{2(q-1/2)^2}(\sigma + b + 1).$$

Proof. Unlike in the proof of Theorem 4.12 we will make use here of the fact that the hypothesis in Theorem 4.9 is only required to hold for sufficiently large λ.

Let $m = \mathsf{prng}(i, b)$. Here we must show that with all but negligible probability, for sufficiently large λ, $x_j \neq x'_j$ implies $e_j := d_2(x_j) \oplus d_2(x'_j) = \mathsf{parity}(\mathsf{andreduce}(x_j \oplus m)) \oplus \mathsf{parity}(\mathsf{andreduce}(x'_j \oplus m))$ is equal to one with probability at least $1 - q$.

We will show that with the choice of $|\mathcal{I}|$ given in the statement, the above will hold with probability $2^{-\sigma}$.

Given $x_j \neq x'_j$, let $y = x_j$ and $y' = x'_j$ to avoid extra subscripts. Let y_l and m_l be the lth bit, with *one* indexing, of y and m respectively. Leaving AND implict (like multiplication) and using $\sum$ to denote XOR, we can rearrange the definition of e_j as follows.

$$e_j = \sum_{l=1}^{b/2} \left(m_{2l-1}\left(y_{2l} \oplus y'_{2l}\right) \oplus m_{2l}\left(y_{2l-1} \oplus y'_{2l-1}\right) \oplus y_{2l}y'_{2l-1} \oplus y_{2l-1}y'_{2l}\right)$$

Let $v(y, y')$ be the vector with entries $y_l \oplus y'_l$ for all $l \in (1, ..., b)$ plus one entry containing $\sum_{l=1}^{b/2} y_{2l}y_{2l-1} \oplus y_{2l-1}y_{2l}$. Note that there are only 2^{b+1} possible values for v.

Now e_j is a function of v and m, we write it as $e(m, v)$. For a fixed value of v let $q(v)$ be the fraction of the key space for which $e(m, v) = 1$. Note that this value has distribution $\text{Bin}(|\mathcal{I}|, 1/2)$ with respect to the randomness of prng. Therefore by a Chernoff bound we have that $\mathbb{P}(q(v) < p) \leq e^{-2(q-1/2)^2|\mathcal{R}|}$. As there are 2^{b+1} possible values of v, a union bound over v yields $\mathbb{P}(\exists v \text{ s.t. } q(v) < p) \leq 2^{b+1}e^{-2(q-1/2)^2|\mathcal{R}|}$. Rearranging, it follows that it suffices to take $|\mathcal{I}| \geq \frac{1}{2(q-1/2)^2}(\sigma + b + 1)$. □

One should think of the σ as a statistical security parameter, thus 40 would be a standard choice.

The prng can be thought of as a fixed function on the set $\mathcal{I}$. We need this function to have the property that any lie v will be caught with probability at least p. This might not need to be a random oracle, but we can not prove any fixed function works, thus we instead show that a randomly selected function works with high probability.

However, it is important here that the randomness for the prng and the parameter b are not chosen adversarially. If they are then the result could still be recovered by increasing σ by however many bits of information about b and the output of prng the adversary was able to control. We will use $\sigma = 40$ when presenting our results.

It would be convenient if given a specific b and prng we could check whether the resulting construction preserves q-collision boundedness. Unfortunately, the problem of determining whether this is the case is as hard as the learning parity with noise problem [33], which is conjectured to be hard.

This does not rule out the idea of replacing the prng with a process that generates an output that is specially structured to guarantee preservation of q-collision boundedness. Indeed this is done in the analogous construction 4 of Section 9 over large fields. However we were unable to find such a construction in the binary case.

The expansion of prng requires $O(b)$ gates, the evaluation of h requires $O(n/b)$ gates, and f requires $n/2$ AND gates and $3n/2$ XOR gates as in construction 2. Thus by taking $b \approx \sqrt{n}$ the total cost is $n/2 + O(\sqrt{n})$ AND gates and $3n/2 + O(\sqrt{n})$ XOR gates.

However, in practice, as $O(n/b)$ is small compared to $n/2$ once b is moderately large we advise taking $b \approx \min(\sqrt{n}, 1024)$ so that for large n the size of $|\mathcal{I}| = O(\sigma + b)$ does not become prohibitive.

The choice of q is somewhat arbitrary but it is a trade-off between wanting something close to $1/2$ whilst not wanting $|\mathcal{I}|$ to be too large. Taking $q = 5/8$ is the compromise we work with.

With $\sigma = 40$, $q = 5/8$ and $b = 1024$ we have $|\mathcal{I}| = 34080$ indices. We will explore these values more in Section 6.

5 PVC COMMITTED MPC FROM INDEXED HASHES

In this section we introduce PVC commitments and the required properties for them to be secure, instantiate them using indexed hash functions, and propose a protocol for committed MPC with PVC security that directly leverages PVC commitments. We will start by defining what a PVC commitment scheme is, then we will explain how to construct one using a collision bounded indexed hash function. We will express the guarantees provided in a theorem and assess how the computational cost of the scheme depends on the indexed hash function. Throughout sk, pk is a public key pair belonging to the committing party that can be thought of as the identity of the input, it should only be used by one input. It is important this public key is associated to the committer (possibly by being signed with another key) by anyone to whom the verifier wishes to prove cheating, e.g. a regulatory authority. The values $i \in \mathcal{I}$ and $r \in \mathcal{R}$ will be randomly chosen as inputs to provide security. For simplicity, we omit the security parameter λ in some of our statements, and when we say that an adversary can not succeed at a task, we mean that they stand a negligible chance of doing so.

5.1 Definitions

We now define PVC commitments in terms of the three functionalities mentioned above.

Definition 5.1. A PVC commitment scheme with covert security parameter $p \in [0, 1]$ consists of three functions pvccommit, assert, check, the last of which is deterministic, satisfying four security properties defined below (correctness, general binding property with parameter p, hiding property, and defamation freeness).

Let us first describe the form of the three functions pvccommit, assert, and check. A commitment function which commits to a value x,

$$c = \text{pvccommit}(x, \text{sk}, r).$$

An assertion function which is applied to the x we later wish to check was committed to,

$$a = \text{assert}(x, \text{sk}, r; i, \text{pk}).$$

And a checking function, which interprets the output from the other two functions,

$$\text{output} = \text{check}(c, a, \text{pk}).$$

With output satisfying $\text{output} \in \{\text{valid}, \text{cheated}, \text{inconclusive}\}$.

Intuitively, output = valid means that the commitment opened to the expected value, output = cheated means that the check did not pass because the committed and asserted values do not match, and output = inconclusive denotes situations where the result of the verification is inconclusive because of a malformed message, or more generally an abort by the committer. This latter situation can not be avoided in general when evaluating PVC commitments in MPC, as a corrupted committer could send invalid messages or stop responding, similar to the role of aborts in MPC security with aborts. The first of the properties is correctness.

Definition 5.2 (Property 1: Correctness). For any i, r, x and valid key pair sk, pk, if $c = \mathsf{pvccommit}(x, \mathsf{sk}, r)$ and $a = \mathsf{assert}(x, \mathsf{sk}, r; i, \mathsf{pk})$ then $\mathsf{check}(c, a, \mathsf{pk}) = \mathsf{valid}$.

The second is binding, a guarantee that a cheating committer will be caught with reasonable probability. P1 can avoid being caught cheating by refusing to sign anything, this is fine so long as they can not possibly get a valid result either. Thus we require that they be caught with probability p only conditioned on the result not being inconclusive. A simple version of this is the following.

Definition 5.3 (Honest Binding). No polynomial time adversary can find $x, \mathsf{sk}, r, x', \mathsf{sk}', r', \mathsf{pk}$ such that (i) $x \neq x'$ and (ii) if $i \leftarrow \mathcal{I}$, $c = \mathsf{pvccommit}(x, \mathsf{sk}, r)$, $a = \mathsf{assert}(x', \mathsf{sk}', r'; i, \mathsf{pk})$ and $\mathsf{output} = \mathsf{check}(c, a, \mathsf{pk})$ then $\mathbb{P}(\mathsf{output} = \mathsf{inconclusive}) < 1$ and

$$\mathbb{P}(\mathsf{output} = \mathsf{cheated} | \mathsf{output} \neq \mathsf{inconclusive}) < p$$

The above allows us to prove PVC security with parameter p only if the commitment is made honestly. If the commitment might be arbitrarily generated then we need the following strictly stronger version of binding. As this version is stronger it is the only one we include in the definition of a PVC commitment scheme, the previous definition will be referenced later in proofs though.

Definition 5.4 (Property 2: General Binding). No polynomial time adversary can find $x, \mathsf{sk}, r, x', \mathsf{sk}', r', \mathsf{pk}$ and c such that (i) $x \neq x'$ and (ii) if $i \leftarrow \mathcal{I}$, $a = \mathsf{assert}(x, \mathsf{sk}, r; i, \mathsf{pk})$, $a' = \mathsf{assert}(x', \mathsf{sk}', r'; i, \mathsf{pk})$, $\mathsf{output} = \mathsf{check}(c, a, \mathsf{pk})$, and $\mathsf{output}' = \mathsf{check}(c, a', \mathsf{pk})$ then $\mathbb{P}(\mathsf{output} = \mathsf{inconclusive}) < 1$, $\mathbb{P}(\mathsf{output}' = \mathsf{inconclusive}) < 1$ and

$$\begin{aligned} &\mathbb{P}(\mathsf{output} = \mathsf{cheated} | \mathsf{output} \neq \mathsf{inconclusive}) \\ &+\mathbb{P}(\mathsf{output}' = \mathsf{cheated} | \mathsf{output}' \neq \mathsf{inconclusive}) < p \end{aligned} \tag{2}$$

To see this is stronger, note that if a scheme is not honestly binding the same counterexample but with $c = \mathsf{pvccommit}(x, \mathsf{sk}, r)$ will show it is not generally binding.

The final two properties prevent the verifier from cheating, so consider sk, pk to be fixed. It is useful to define an oracle $O_{\mathsf{sk}}(x)$ which when called samples $r \leftarrow \mathcal{R}$ and returns

$$\mathsf{pvccommit}(x, \mathsf{sk}, r)$$

and

$$(\mathsf{assert}(x, \mathsf{sk}, r; i, \mathsf{pk}))_{i \in \mathcal{I}}.$$

The third property is the hiding property which guarantees the verifier can not learn anything about x from the outputs of pvccommit or assert.

Definition 5.5 (Property 3: Hiding). For any x, x' and polynomial time adversary $\mathcal{A}$

$$\mathbb{P}(\mathcal{A}(O_{\mathsf{sk}}(x)) = 1) = \mathbb{P}(\mathcal{A}(O_{\mathsf{sk}}(x')) = 1) + \mathrm{negl}(\lambda).$$

The final property is defamation freeness which guarantees the verifier can not frame an honest committer.

Definition 5.6 (Property 4: Defamation Freeness). No polynomial time adversary can choose an x and then when given $O_{\mathsf{sk}}(x)$ find c and a such that

$$\mathsf{check}(c, a, \mathsf{pk}) = \mathsf{cheated}$$

Note it is important that each secret key is only used for one choice of x, r. This could be achieved by deriving the secret key from (x, r) by a one way function (possibly with extra randomness).

5.2 PVC commitment from indexed hashes

Let H be an indexed hash function with index space $\mathcal{I}$ and randomness space $\mathcal{R}$. Let $m_{\mathrm{sgn}(\mathsf{sk})}$ denote m together with a signature of m by secret key sk. Consider the following three functions.

$$\mathsf{pvccommit}(x, \mathsf{sk}, r) = ((H(i, r, x))_{i \in \mathcal{I}})_{\mathrm{sgn}(\mathsf{sk})}$$

$$\mathsf{assert}(x, \mathsf{sk}, r; i, \mathsf{pk}) = \begin{cases} (i, H(i, r, x))_{\mathrm{sgn}(\mathsf{sk})} & \text{if } (\mathsf{sk}, \mathsf{pk}) \text{ is a valid keypair} \\ \perp & \text{otherwise} \end{cases}$$

For check let G be the event that the signatures are valid.

$$\mathsf{check}(c, a, \mathsf{pk}) = \begin{cases} \mathsf{valid} & \text{if } G \text{ and } c[a[0]] = a[1] \\ \mathsf{cheated} & \text{if } G \text{ and } c[a[0]] \neq a[1] \\ \mathsf{inconclusive} & \text{Otherwise} \end{cases}$$

We require one slightly unusual property of the signature scheme. This is a technicality, as (a) lots of schemes have this property and (b) in the next subsection we will introduce a computational optimization which has the side effect of guaranteeing this property from any scheme.

Definition 5.7. Call a signature scheme *discrimination resistant* if no polynomial time adversary can find m, m', sk and pk, ((sk, pk) not necessarily a valid key pair), such that $m_{\mathrm{sgn}(\mathsf{sk})}$ and $m'_{\mathrm{sgn}(\mathsf{sk})}$ are valid with non-negligibly different probabilities.

We also require that the signature scheme has a deterministic verification function. This could be lifted at the expense of complicating the definitions with extra negligible terms. However, whilst not implied by the definition of a signature scheme, all the most popular schemes satisfy this assumption so we will make it for simplicity.

Theorem 5.8. *If H is hiding and q-collision bounded and the signature scheme has deterministic verification and is discrimination resistant, then the above functions form a PVC commitment scheme with covert security parameter $p = 1 - q$ (Definition 5.1).*

The proof of this theorem is given in Appendix E.1.

5.3 PVC Committed MPC from a PVC commitment scheme

In this section we define formally PVC committed MPC, for the two party case, and propose protocols to efficiently realize this functionality, which corresponds to the intuitive idea from Figure 1.

We follow the definitions by Asharov and Orlandi [5] to prove PVC security of our protocols. This involves proving (i) simulatability (in the ideal vs. real worlds framework) for the covert security part, along with (ii) accountability and (iii) defamation freeness for the public verifiability. For (ii) and (iii) we use the definitions by Asharov and Orlandi and for (i) our ideal world is presented in detail in Appendix E.2 as an extension of theirs, to handle the commitment phase. Without loss of generality, we describe our ideal world for only two parties P1 and P2. Moreover, as in our protocols, the first party gets malicious security, while the second party gets PVC security. This matches the guarantee in the generic PVC protocol by Hong et al [23] that we use in the experimental evaluation.

Our ideal world is parameterized by two values $p_{\mathsf{exec}}, p_{\mathsf{commit}} \in [0,1]$ denoting lower bounds on the probabilities with which P1 can get caught when (i) cheating in the protocol execution and (ii) breaking the commitment, respectively. Note that Asharov and Orlandi only formalize (i), and they denote p_{exec} as ϵ. Moreover, our ideal world is parametrized by an arbitrary distribution $\mathcal{E}$ with we refer to as *the environment* (this is similar to the notion used in the UC framework). A sample from the environment is included in the parties' view as an auxiliary input that is received only *after* the commitment phase has finished. This limits the ability of the ideal world adversary (the simulator) to rewind the adversary beyond the commitment phase (similar to the role of the environment in UC), and models information that the adversary might get after committing.

We summarize the ideal world execution next. First, P1 receives its prescribed input and commits to it (if honest) or an arbitrary value (if corrupted) by sending it to the trusted party. This constitutes the commitment phase, and captures the situation where P1 commits to using an input, e.g., an ML model, to be used at an undetermined time in the future in a secure computation with a second party P2. Then, party P1 receives an input from the environment, in the form of a sample from $\mathcal{E}$, which is also given to P1 in the real world, as explained above. This determines the beginning of the secure computation phase, which starts with P2 receiving its prescribed input and with P1 notifying the trusted party of their desire to cheat in the execution. This attempt will succeed with probability $1 - p_{\mathsf{exec}}$, in which case P1 gets to completely break the protocol, i.e. learn P2's input and choose their output. If P1 fails, P2 receives output `corrupted`. If a corrupted P1 decided to not cheat in this way, they still get a chance to cheat in switching the input of the secure computation from the committed value w to a different one. If this attempt fails (which happens with probability at least p_{commit}), P2 gets notified. For simplicity in the presentation we allow P1 to abort after receiving their output, and before P2 gets to observe theirs, but this assumption can be lifted by ensuring that in the underlying PVC protocol P2 gets the output first.

Public Parameters: A PVC commitment scheme (Definition 5.1) and a public key pk.
Inputs: input x and secret key sk matching pk.
Outputs: Commitment c.

Algorithm:
(1) Sample $r \leftarrow \mathcal{R}$.
(2) Compute $c = \mathsf{pvccommit}(x, \mathsf{sk}, r)$.
(3) Store r as a secret and return c.

Figure 3: PVC Committed 2PC (commitment algorithm).

Let (pvccommit, assert, check) be a PVC commitment scheme with parameter p. Let $\mathtt{Blame}_{\mathsf{commit}}$ be the function which when given a view of P2 (honestly) running the protocol in Fig. 4, in which output = cheated returns the commitment c and the resulting a and otherwise returns $\perp$. Let $\mathtt{Judgement}_{\mathsf{commit}}$ be the function check with the public key of P1 hard coded. Let `Commit` be the commitment algorithm in Fig. 3 and $\mathcal{P}$ be the protocol in

Parties: P1, P2.
Public Parameters: A PVC commitment scheme (Definition 5.1), a commitment c, and a public key pk.

The protocol uses a PVC secure protocol Π offering PVC security to P2 and malicious security to P1.
Inputs: P1: x, r; P2: y.
Outputs: P1 : $g_1(x, y)$; P2 : $g_2(x, y)$, or a proof of cheating a.

Protocol:
(1) P2 samples $i \leftarrow \mathcal{I}$.
(2) P1, P2 run Π to compute
$(o_1; o_2, a) = (g_1(x, y); g_2(x, y), \mathsf{assert}(x, \mathsf{sk}, r; i, \mathsf{pk}))$.
(3) P2 computes output = $\mathsf{check}(c, a, \mathsf{pk})$ and
If output = valid $\longrightarrow$ accepts o_2 as $g_2(x, y)$.
If output = cheated $\longrightarrow$ accepts a as proof of cheating.
Otherwise $\longrightarrow$ aborts and sets result to *inconclusive*.

Figure 4: PVC Committed 2PC for functionality $g(x, y) = (g_1(x, y), g_2(x, y))$ (integrity check).

Parties, inputs, outputs, and public parameters are as in Figure 4, and the PVC commitment scheme is instantiated by an indexed hash function H (as in Theorem 5.8).

Protocol:
(1) P2 samples $i \leftarrow \mathcal{I}$ and $\tilde{r} \leftarrow \mathcal{R}$.
(2) P1, P2 run Π to compute $(g_1(x, y), h(m|\tilde{r}); g_2(x, y), m)$, where $m = (i, H(i, r, x))$.
(3) P1 computes $s = \mathsf{sign}(h(m|\tilde{r}), \mathsf{sk})$ and sends it to P2.
(4) P2 aborts if s is not the valid signature of $h(m|\tilde{r})$.
(5) P2 computes output = $\mathsf{check}(c, a, \mathsf{pk})$ and
If output = valid $\longrightarrow$ accepts o_2 as $g_2(x, y)$.
If output = cheated $\longrightarrow$ accepts a as proof of cheating.
Otherwise $\longrightarrow$ aborts and sets result to *inconclusive*.

Figure 5: PVC Committed 2PC for functionality $g(x, y) = (g_1(x, y), g_2(x, y))$ (Optimized integrity check).

Fig. 4, with Π instantiated with the protocol of Hong et al [23]. Finally, let $\mathtt{Blame}_{\mathsf{exec}}$ and $\mathtt{Judgement}_{\mathsf{exec}}$ be the blame and judgement functions from Π, and define $\mathtt{Blame}(x)$ to be cheated if either $\mathtt{Blame}_{\mathsf{exec}}(x)$ or $\mathtt{Blame}_{\mathsf{commits}}(x)$ equals cheated, and analogously for a function `Judgement`. We are now ready to state our main result.

Theorem 5.9. *The quadruple* $(\mathtt{Commit}, \mathcal{P}, \mathtt{Blame}, \mathtt{Judgement})$ *securely computes g with committed first input in the presence of a malicious* P1 *or a covert* P2 *with $p/2$-deterrent and public verifiability.*

If pvccommit *and* check *are used as given in the previous section then we can replace the $\mathcal{P}$ with the protocol in Fig. 5 and still have the same security guarantee.*

Furthermore, if in either case it can be guaranteed that P1 *is honest in running the commitment algorithm in Fig. 3, then the deterrent factor improves from $p/2$ to p.*

Non-committed output at no risk. P1 can in the above ideal world, and thus in the protocol, get $g_1(x', y)$ for a non-committed x' at no risk by aborting afterwards. This could be avoided by opening up the PVC blackbox and holding back this output until P2 has checked the result of assert (or optimized equivalent).

Computational costs. The cost of the commit operation in the clear is computing H, $|\mathcal{I}|$ times. The cost of the assert is dominated asymptotically by the cost of computing H once i.e. requires $n/2 + o(n)$ AND gates. The check are $O(1)$ and relatively very cheap.

6 EVALUATION

Here we compare our method for committed MPC to the baseline using SHA3-256. We evaluate both computation time and communication for the assert functionality as the size of the input n increases. We also analytically compare our method against the hash function based on LowMCHash-256, an MPC friendly hash [3]. Finally, we evaluate the practicality of our proposed scheme in terms of the compute requirement for the committer performing the commitment using the pvccommit functionality and the size of the commitment. As a result, we show, for our scheme: (a) Verification (assert functionality) in MPC is significantly faster than optimized standards such as SHA3-256 as well as MPC optimized hashes such as LowMCHash-256; (b) The size of the commitment is practical; (c) The computation required from the committer (pvccommit functionality) is practical. We use the circuit sizes and real experimental data for (a). Similarly, we analyze the size of the commitment to prove (b) and use actual computation time data to show (c). We begin by describing the experimental and implementation details.

Experimental Settings. The experiments were executed on two Azure D32s v3 machines running Ubuntu 16.04, equipped with Intel Xeon E5-2673 v4 2.3GHz processors and 128 GB RAM. The machines were hosted in the same region with a bandwidth of 1.7 GB/s and an avg. latency of 0.9ms, representative of a LAN setting.

Implementation. We use the EMP-toolkit [39] to implement our secure protocols as well as the baselines. In particular, we use the PVC framework of Hong et al. [21], which makes use of garbled circuits. We set the covert security parameter p_c of this underlying implementation to $1/2$. Note this is different from the covert security parameter p used in our scheme. Since $p \leq 1/2$, p_c could be set to $1/2$. As one could infer, the effective covert security parameter for our scheme with this implementation would be $min(p, p_c)$.

Baselines. We use two baselines for comparison: SHA3-256 and LowMCHash-256. For SHA3-256, we use the sponge framework [9] with an input block size of 1600. Using the standard security parameters we get the rate as 1088 and the capacity as 512. This results in a computation cost of $\sim$ 35 AND gates per input bit. For LowMCHash-256, we use LowMC permutations together with the sponge framework using an input block size of 512. We reserve 256 bits for the rate and another 256 bits for the capacity (128 bit security). This results in $\sim$ 14 AND gates per input bit. LowMC is relatively new and has been shown to be susceptible to attacks [14]. However, we include it in this comparison, it being one of the most MPC optimized hashing schemes for Boolean circuits. Both these baselines are implemented on the top of the EMP-toolkit's PVC framework.

Our scheme. For our scheme we implement the idea around PVC commitment from indexed hashes as described previously. We use Construction 3 in section 4. In particular we implement the assert functionality in MPC (and pvccommit, check in the clear). Our scheme costs $\sim$ 0.5 AND gate per input bit. In order to effect signed public verifiability during the assert phase, we use SHA3-256 to commit the hash corresponding to the input and the index. We summarise the parameters for our scheme in Table 3. Note that the table reports the covert security parameter of the commitment scheme for the honest committer case. In the general case this parameter's value would be $p/2 = 3/16$. Similar to the baselines, our scheme is also implemented on top of the EMP-toolkit's PVC framework.

6.1 Analytical Performance

Table 2 compares the circuit size $|C|$ (no. of AND gates) for the assert functionality for the LowMCHash-256 and SHA3-256 baselines with our scheme. As we increase the size of the input, the scheme starts to show its full potential. For a small input size, the initial overhead of signing the commitments and the index tends to shadow the improvement. But as we increase the size of the input, we can see a marked 70× improvement over SHA3-256 and 28× improvement over LowMCHash-256. We show that these improvements directly translate into real world experiments, when compared against the actual implementation of SHA3-256, in section 6.2.

6.2 Experimental Performance

Running time for assert**.** Table 4 shows the running time for executing the assert functionality to verify the commitments using SHA3-256 and our scheme. As we increase the size of the input to practical sizes, we observe that our scheme is 60× faster than the SHA3-256 baseline. This is directly correlated with the 70× improvement in the circuit sizes above. We do not perform actual experiments with LowMCHash-256, but it is similarly expected to be around 25× slower than our scheme as indicated by the circuit sizes. Also, in practice, nothing prohibits us from replacing our underlying collision resistant hash h with LowMCHash to amplify this improvement. We compare the

Communication for assert**.** Table 6 (Appendix B) shows the amount of communication needed for executing the assert functionality using SHA3-256 and our scheme. We observe that our scheme requires 36× less communication for the committer and the verifier than the SHA3-256 based baseline.

Computation load for pvccommit**.** In Table 5, we show the number of indices $|\mathcal{I}|$ for the commitment that needs to be computed alongside the size of the entire commitment that a committer needs to prepare in order to commit its input. In these experiments $p = 3/8$ ($q = 5/8$) and block size $b = min(\sqrt{n}, 1024)$. The size of the commitment results in a very limited communication and space requirement. Block size limit of 1024 bits, limits the size of the commitment to just 1.09 MB. We use the formulation, upon ceiling to the next nearest integer, defined in Theorem 4.13 to compute $|\mathcal{I}|$. In Figure 6 (Appendix C), we plot this formulation for $\sigma = 40$, $b = 1024$ and different values of q (and the covert security parameter p i.e $1 - q$)

	SHA3-256	LowMCHash-256	Ours		
No. of bits	# of ANDs	# of ANDs	# of ANDs	Improvement over SHA3-256	Improvement over LowMCHash-256
2^{14}	6.14×10^5	2.32×10^5	5.17×10^4	12×	4×
2^{18}	9.29×10^6	3.65×10^6	1.90×10^5	49×	19×
2^{22}	1.48×10^8	5.84×10^7	2.29×10^6	65×	25×
2^{26}	2.37×10^9	9.34×10^8	3.42×10^7	69×	27×
2^{30}	3.79×10^{10}	1.49×10^{10}	5.39×10^8	70×	28×

Table 2: Analytical comparison of the number of AND gates (circuit size $|C|$) for the assert **functionality using LowMCHash-256 and SHA3-256 with our scheme. These values are for a single call to** assert **i.e. using a single index for our scheme. Here $p_c = 1/2$**

Description	Symbol	Value
Our Indexed Hash Function		
Length of the input ($|x|$)	n	no. of bits (variable)
Pseudorandom number generator	$prng$	AES (counter mode)
Underlying collision resistant hash	h	SHA3-256
Statistical security parameter	σ	40
Collision boundedness parameter	q	5/8
Block size	b	$min(\sqrt{n}, 1024)$
Our PVC Commitment		
Covert security parameter	p	$1 - q = 3/8$
Underlying PVC 2PC Protocol (EMP-PVC) For assert		
Covert security parameter	p_c	1/2

Table 3: Parameters used in our experiments

No. of bits	Ours (s)	SHA3-256 (s)	Improvement
2^{14}	0.07	0.57	8×
2^{18}	0.22	8.16	36×
2^{22}	2.67	133.23	50×
2^{26}	39.14	2200*	56×
2^{30}	590.70	35500*	60×

Table 4: Comparison of running time for SHA3-256 baseline and our scheme executing the assert **functionality. Here $p_c = 1/2$. * means estimated via extrapolation**

to show how the number times $|\mathcal{I}|$ that the committer needs to compute H varies with the security parameters.

In Table 5 we also show the computation load of the committer for committing its input. In particular, we evaluate the time needed to perform the pvccommit functionality. This only needs to be performed once for a given input, in the clear. We see that the computation load is very limited even for large input sizes. For these results we use only a single process, however this computation is trivially parallelizable. Several hashes can be computed in parallel. For example a 128 threaded implementation should enable the committer to commit 2^{30} bits in less than 3 minutes. Furthermore, we perform these computations in Python using standard libraries

No. of bits	# of Hashes	Size of the Commitment (MB)	Time
2^{14}	5408	0.17	1.61s
2^{18}	17696	0.57	28.21s
2^{22}	34080	1.09	5.06m
2^{26}	34080	1.09	36.92m
2^{30}	34080	1.09	5.91h

Table 5: Computation time and size of the commitment using our scheme for executing the pvccommit **functionality. Here $p = 3/8$, statistical security parameter $\sigma = 40$ and block size $b = min(\sqrt{n}, 1024)$ where n is the size of input.**

and there is scope for further significant optimization by using a low-level language.

7 LOWER BOUNDS

In this section we provide lower bounds on how many AND gates are required for a collision resistant indexed hash function and an ordinary hash function. Recall that our construction 3 from Section 4 requires half an AND gate per bit of input. In this section we show that

(1) Construction 3 is optimal amongst hash functions whose output's size is sublinear w.r.t. their input's size (Corollary 7.2).
(2) For ordinary hash functions we show that every collision resistant hash function requires at least one AND gate per input bit (Proposition 7.3).
(3) Assuming that we want our hash functions to be hiding, we show, both for indexed and ordinary hash functions, that allowing their output to be large does not help much to reduce the number of required nonlinear gates (Proposition 7.4).

Moreover, although we state the above results in terms of Boolean circuits, it is not hard to see that the arguments extend to any field. The following lemma and corollary correspond to item 1 above. The proof, given in Appendix F, constructs an algorithm to find a 1-collision on any H with small set of nonlinear gates by casting that problem as that of solving a linear system S on $\mathbb{F}_2$, and showing that S always has a solution. Recall that indexed hash functions have three inputs i, r, x, in the statement by *main input* we mean x.

Proposition 7.1. *Given any non-trivially collision bounded family of indexed hash functions $\{H_k\}_{k\in K}$ with H_k given by the (polynomial size) circuit C_k with n-bit main input, and m-bit output. With all but negligible probability over the generation of $k = G(\lambda)$, the circuit C_k must have at least $\lceil (n-m)/2 \rceil$ nonlinear gates.*

Note that in practice the lower bound on the nonlinear gate count will apply (with all but negligible probability) for any λ large enough to be considered secure. In particular we have the following corollary which says that, in order to beat our constructions asymptotically, an indexed hash function must have large output.

Corollary 7.2. *Any family of covertly collision resistant hash function circuits, indexed by n, with main input in $\{0,1\}^n$ must either have at least $n/2 + o(n)$ nonlinear gates or must have output size that is not $o(n)$.*

A stronger result can be achieved in the case of an ordinary secure hash function, by relying on the fact that they do not take auxiliary inputs. The idea of the proof is similar to that of Proposition 7.1, and is given in Appendix F.

Proposition 7.3. *Let $\{h_k\}_{k\in K}$ be a collision resistant family of hash functions with h_k given by the circuit C_k with n-bit input and m-bit output. With all but negligible probability with respect to the generation of $k = G(\lambda)$, the circuit C_k must have at least $n - m$ nonlinear gates.*

These results show that our constructions have asymptotically half the verification cost of the baseline with any ordinary secure hash function. However, recall that we designed our construction 3 for the output of H to be small, i.e. $o(n)$, for *efficiency and not security reasons*. One may thus wonder whether dropping this requirement allows to significantly overcome the above lower bounds. Next, we show that the answer is negative by leveraging the fact that we do require a hiding property *for security*, which we show implies a linear lower bound on the required number of AND gates.

The proof of the following result can be found in Appendix F. It relies on the fact that if you have a small number of AND gates then only a small amount of the entropy in the randomness can affect their inputs. The rest of the randomness can not be used for hiding the output without giving too much leeway for finding collisions. Thus only a small amount of randomness and a small number of output wires from AND gates can hide the output. Thus the output must be effectively small and the above propositions can be applied.

Proposition 7.4. *Suppose that $\{H_k\}_{k\in K}$ is a non-trivially collision bounded and hiding family of hash functions. Let H_k be given by C_k with an n-bit main input and d nonlinear gates, then with all but negligible probability, $d \geq n/5$. Further if $|\mathcal{I}| = 1$, then $d \geq n/3$.*

8 FROM COVERT TO MALICIOUS SECURITY

A natural idea is to amplify the statistical guarantee of an indexed hash function H by computing it at several indices. This would in turn lead to a PVC commitment scheme with improved parameters where H is run on several indices. More concretely, given a collision resistant indexed hash function H we can compute an indexed hash function H^κ with stronger security by computing H κ times with different indices. Formally, with $i_j \in \mathcal{I}$ for $j \in \{1, ..., \kappa\}$

$$H^\kappa((i_j)_{j=1}^\kappa, r, x) = \left(H(i_j, r, x)\right)_{i=1}^\kappa.$$

This new function requires no more hashes to be prepared by the committer and, if $\{H_k\}_{k\in K}$ is q-collision bounded then $\{H_k^\kappa\}_{k\in K}$ is q^κ-collision bounded. However, it also requires κ times as many AND gates (and XOR gates) to compute it. In this section, we present a construction that asymptotically requires no more AND gates than H (and fewer XOR gates than H^κ) to achieve this higher security.

Let $E : \{0,1\}^w \to \{0,1\}^l$ be the encoding function of a $(\kappa - 1)$ error detecting code. All we require from E is that if two messages $m, m' \in \{0,1\}^w$ then their codes, i.e. $E(m), E(m') \in \{0,1\}^l$ differ in at least κ positions. Split x into w words, $x_1, ..., x_w$ each of length $\lceil n/w \rceil$, zero-padding x as required. Let $x^1, ..., x^{\lceil n/w \rceil}$ be the columns of the matrix whose rows are given by the x_j. Let $\tilde{x}_1, ..., \tilde{x}_l$ be the rows of the matrix whose columns are given by $E(x^1), ..., E(x^{\lceil n/w \rceil})$. Finally let

$$H^E\left((i_j)_{j=1}^l, r, x\right) = \left(H(i_j, r, \tilde{x}_j)\right)_{j=1}^l.$$

The following theorem follows from the structure of H^E and the property of the error detecting code (proof in Appendix G).

Theorem 8.1. *If $\{H_k\}_{k\in K}$ is q-collision bounded then $\{H_k^E\}_{k\in K}$ is q^κ-collision bounded.*

Furthermore, the number of AND and XOR gates required to compute H_E is $l\lceil n/w \rceil$ times the number of gates required per bit by H plus $\lceil n/w \rceil$ times the number of gates required by E.

To make use of the above result we need an error detecting code E that works on fairly large codewords and is easy to compute. We want it to be linear to keep the number of AND gates low, but we also do not want to introduce too many XOR gates. The following lemma provides such an encoding.

Lemma 8.2. *Given $\rho, d \in \mathbb{Z}_+$, there exists a linear $2^d - 1$ error detecting encoding $E : \{0,1\}^{\rho^d} \to \{0,1\}^{(\rho+1)^d}$ requiring $(\rho-1)((\rho+1)^d - \rho^d)$ XOR gates to compute.*

Proof. Given a message $m \in \{0,1\}^{\rho^d}$, arrange the bits of m in a d-dimensional cube. We index into m with the notation $m[i_1, ..., i_d]$. We extend m by one in each dimension in turn by the following method. To extend m by one in the dimension j, let $m[i_1, ..., i_{j-1}, \rho, i_{j+1}, i_d]$ be the XOR of $m[i_1, ..., i_{j-1}, 0, i_{j+1}, i_d]$ through $m[i_1, ..., i_{j-1}, \rho - 1, i_{j+1}, i_d]$. The output of E is just the contents of the resulting cube.

Let m' be a different message, then for some choices of i_j we have that $m[i_1, ..., i_d] \neq m'[i_1, ..., i_d]$. We can then deduce by induction that after j dimensions have been extended there are at least 2^j points in the cuboids with final co-ordinates $i_{j+1}, ..., i_d$ on which m and m' differ. Thus once all directions have been extended the arrays m and m' differ in at least 2^d places and we have a $2^d - 1$ error detecting code.

The jth extension requires $(\rho - 1)(\rho + 1)^{j-1}\rho^{d-j}$ XOR gates. Summing over all j gives the result. □

Putting the above together we get a corollary which says there exists an asymptotically efficient protocol for maliciously secure commitment. Note that $\log 1/q$ is a statistical security parameter so can be thought of as a small constant, independent of n and λ, in practice $\log_2 \log_2 1/q = 6$ should suffice.

Corollary 8.3. *Assume the existence of a collision resistant family of hash functions $\{h_k\}_{k\in K}$ with run time linear in input size and*

a random oracle prng. *Then there exists a q-collision bounded indexed hash function family with the following two properties. For a fixed security parameter, it can be computed with $n/2 + o(n \log\log 1/q)$ AND gates and $(5/2 + \lceil \log_2 \log_2 1/q \rceil)n + o(n \log\log 1/q)$ XOR gates. It requires $o(n \log_2 \log_2 1/q)$ information to be stored in order to be able to check any result.*

Furthermore, if $\{h_k\}_{k\in K}$ is replaced by a family of random oracles then the resulting indexed hash function family is hiding.

Proof. Let E be the encoding function given in Lemma 8.2 with $d = 1 + \lceil \log_2 \log_2 1/q \rceil$ and $\rho = \lceil n^{1/3d} \rceil$. Let $H = C_3(h_k)$ with $|\mathcal{I}|$ chosen to give collision resistance with parameter $1 - \sqrt{1/2}$. Then $\{H^E_k\}_{k\in K}$ has all the required properties. □

We have not done any experiments with this idea, however from preliminary estimates of AND gate counts (with $q = 2^{-2^6}$) we are confident that it offers no improvement for inputs of 10^6 bits. If the choices of parameters were optimized we believe it would beat the baselines for $n = 10^9$, though the cross over point depends on the baseline and choice of h (and prng).

This effectively recovers malicious security in the setting where the commitment is honestly generated, by the results of Section 5. In fact, however, this method can recover malicious security in the presence of arbitrarily generated commitments too. As on all but at most one input (decided at commitment time) H will catch cheating with probability $p/2$, it can be guaranteed that H^E will catch cheating with all but probability $(1 - p/2)^\kappa$. Thus for the not honestly committed case we need to only increase the choice of d by one in the proof of Corollary 8.3.

9 ARITHMETIC CIRCUITS

We have mainly focused on binary circuits because they are more flexible and there are more reasonably fast hash functions for them. However our main idea will also work to construct indexed hash functions to be computed in arithmetic circuits. As before our constructions are in terms of a secure hash function h which could be implemented using MiMC [2] or any other arithmetic circuit hash function. We will assume this arithmetic is in a field $\mathbb{F}$.

Analogues of constructions 2 and 3 would work in this setting with XOR and AND gates replaced by ADD and MUL gates. Indeed, these would also work, with worse parameters, over arbitrary rings. These can be analysed analogously and relevant theorems deduced. However we will not detail these changes here and will instead provide a further development that was not possible in the binary case.

The idea of construction 4 presented in this section is much like the analogue of construction 3, however instead of using a prng to generate the random masks to be added to index, we will generate them in a more structured fashion. Hence, construction 4 still follows the blueprint given in Equation 1. The index space $\mathcal{I}$ will be a subset of $\mathbb{F}$, this requires the field to be moderately large and rules out this construction in the binary case.

As in Section 4, we have an even block size parameter b, and define the indexed hash by means of a digest function d_4 that takes an index i and b field elements as input and returns a single field element. Given y a fixed block of b elements denoted by $y_1, ..., y_b$,

$$d_4(i, y) = \sum_{j=1}^{b/2} (i^{2j-1} + y_{2j-1})(i^{2j} + y_{2j})$$

The value of the hash is given by $C_4(h, \lambda)(i, r, x) = h(r||i||\texttt{process}_{b,d_4)}(i, x))$ given functions process and h, as described in Equation 1 and Section 4.

The following Theorem states the guarantee of construction 4. While its full proof is given in Appendix H, the basic idea is that there will be a collision so long as some vector determined from x and x' is not perpendicular to $(1, i, i^2, ..., i^b)$. The powers of i come from the definition of d_4 and have been chosen (to replace the prng) so that these vectors form Vandermonde matrices, thus any $b+1$ of them span and so at most b are perpendicular to any given vector.

Theorem 9.1. *Construction 4 is $b/|\mathcal{I}|$-collision bounded.*

Note that this construction only works for fields larger than the block size b, but this is the case for a lot of standard hashes based in field arithmetic. If the field is very large then the covert security parameter can be made ≈ 1 by taking $|\mathcal{I}|$ to be big. However this would be very impractical to prepare the hashes, and thus in that case it would be more practical to combine construction 4 presented in this section with the amplification ideas from section 8.

10 CONCLUSION

The standard simulation-based security definitions used in MPC allow a malicious adversary controlling one of the parties to provide arbitrary inputs. This leaves concerns related with input validity. In this paper, we introduced a method for securely committing an input in 2PC publicly verifiable covert (PVC) model for Boolean circuits. PVC security is valuable when the reputation of the committing party is at stake. Our methods are based on our introduction of indexed hashes and $q-$collision resistance and make use of the covert security guarantees and interactivity in MPC. Our work improves upon ordinary hash functions both in speed and communication. Our work is the first we are aware of to enable commitments in MPC for PVC security. We also extend our methods to the maliciously secure model and arithmetic circuits.

Future work could evaluate these methods for certified prediction and for the maliciously secure variant with optimized parameters. There is also a gap to be closed between constructions and lower bounds if we allow large commitments. The requirements on prng are slightly inconvenient and a deterministic way to find vectors for construction 3, like in construction 4, would be useful. One could investigate if the commitment size could be reduced to $O(1)$ while maintaining half an AND gate per bit cost.

ACKNOWLEDGMENTS

NA was supported by University of Oxford and Callsign. This work was done when AG was at The Alan Turing Institute (ATI) and Warwick University. AG and JB were supported by ATI under the EPSRC grant EP/N510129/1, and the UK Government's Defence & Security Programme. We also acknowledge ATI's support and generous provision of Azure cloud computing resources.

REFERENCES

[1] Nitin Agrawal, Ali Shahin Shamsabadi, Matt J Kusner, and Adrià Gascón. Quotient: two-party secure neural network training and prediction. In *Proceedings*

of the 2019 ACM SIGSAC Conference on Computer and Communications Security, pages 1231–1247, 2019.

[2] Martin Albrecht, Lorenzo Grassi, Christian Rechberger, Arnab Roy, and Tyge Tiessen. Mimc: Efficient encryption and cryptographic hashing with minimal multiplicative complexity. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 191–219. Springer, 2016.

[3] Martin R Albrecht, Christian Rechberger, Thomas Schneider, Tyge Tiessen, and Michael Zohner. Ciphers for mpc and fhe. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 430–454. Springer, 2015.

[4] Dario Amodei, Chris Olah, Jacob Steinhardt, Paul Christiano, John Schulman, and Dan Mané. Concrete problems in ai safety. *arXiv preprint arXiv:1606.06565*, 2016.

[5] Gilad Asharov and Claudio Orlandi. Calling out cheaters: Covert security with public verifiability. In *ASIACRYPT*, volume 7658 of *Lecture Notes in Computer Science*, pages 681–698. Springer, 2012.

[6] Gilad Asharov, Yehuda Lindell, Thomas Schneider, and Michael Zohner. More efficient oblivious transfer and extensions for faster secure computation. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 535–548. ACM, 2013.

[7] Yonatan Aumann and Yehuda Lindell. Security against covert adversaries: Efficient protocols for realistic adversaries. *J. Cryptology*, 23(2):281–343, 2010.

[8] Aner Ben-Efraim, Yehuda Lindell, and Eran Omri. Efficient scalable constant-round mpc via garbled circuits. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 471–498. Springer, 2017.

[9] Guido Bertoni, Joan Daemen, Michael Peeters, and Gilles Van Assche. On the indifferentiability of the sponge construction. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 181–197. Springer, 2008.

[10] Mariusz Bojarski, Philip Yeres, Anna Choromanska, Krzysztof Choromanski, Bernhard Firner, Lawrence Jackel, and Urs Muller. Explaining how a deep neural network trained with end-to-end learning steers a car. *arXiv preprint arXiv:1704.07911*, 2017.

[11] Ran Canetti and Marc Fischlin. Universally composable commitments. In Joe Kilian, editor, *Advances in Cryptology — CRYPTO 2001*, pages 19–40, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg. ISBN 978-3-540-44647-7.

[12] Ivan Damgård, Martin Geisler, and Jesper Buus Nielsen. From passive to covert security at low cost. In *TCC*, volume 5978 of *Lecture Notes in Computer Science*, pages 128–145. Springer, 2010.

[13] Thomas Davenport and Ravi Kalakota. The potential for artificial intelligence in healthcare. *Future healthcare journal*, 6(2):94, 2019.

[14] Itai Dinur, Yunwen Liu, Willi Meier, and Qingju Wang. Optimized interpolation attacks on lowmc. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 535–560. Springer, 2015.

[15] Tore K Frederiksen, Benny Pinkas, and Avishay Yanai. Committed mpc. In *IACR International Workshop on Public Key Cryptography*, pages 587–619. Springer, 2018.

[16] Adrià Gascón, Phillipp Schoppmann, Borja Balle, Mariana Raykova, Jack Doerner, Samee Zahur, and David Evans. Privacy-preserving distributed linear regression on high-dimensional data. *Proceedings on Privacy Enhancing Technologies*, 2017 (4):345–364, 2017. doi: 10.1515/popets-2017-0053.

[17] Zahra Ghodsi, Tianyu Gu, and Siddharth Garg. Safetynets: Verifiable execution of deep neural networks on an untrusted cloud. In *Advances in Neural Information Processing Systems*, pages 4672–4681, 2017.

[18] Ran Gilad-Bachrach, Nathan Dowlin, Kim Laine, Kristin E. Lauter, Michael Naehrig, and John Wernsing. Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy. In *ICML*, volume 48 of *JMLR Workshop and Conference Proceedings*, pages 201–210. JMLR.org, 2016.

[19] Vipul Goyal, Payman Mohassel, and Adam D. Smith. Efficient two party and multi party computation against covert adversaries. In *EUROCRYPT*, volume 4965 of *Lecture Notes in Computer Science*, pages 289–306. Springer, 2008.

[20] Zecheng He, Tianwei Zhang, and Ruby B Lee. Verideep: Verifying integrity of deep neural networks through sensitive-sample fingerprinting. *arXiv preprint arXiv:1808.03277*, 2018.

[21] Cheng Hong, Jonathan Katz, Vladimir Kolesnikov, Wen-jie Lu, and Xiao Wang. Covert security with public verifiability: Faster, leaner, and simpler. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 97–121. Springer, 2019.

[22] Yuval Ishai, Eyal Kushilevitz, and Rafail Ostrovsky. Sufficient conditions for collision-resistant hashing. In Joe Kilian, editor, *Theory of Cryptography*, pages 445–456, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg. ISBN 978-3-540-30576-7.

[23] Cheng Hong Jonathan Katz Vladimir Kolesnikov Wen jie Lu Xiao Wang. Covert security with public verifiability: Faster, leaner, and simpler. In *EuroCrypt*, 2019.

[24] Niki Kilbertus, Adria Gascon, Matt Kusner, Michael Veale, Krishna P Gummadi, and Adrian Weller. Blind justice: Fairness with encrypted sensitive attributes. In *International Conference on Machine Learning*, pages 2635–2644, 2018.

[25] Yehuda Lindell. Fast cut-and-choose based protocols for malicious and covert adversaries. In *CRYPTO (2)*, volume 8043 of *Lecture Notes in Computer Science*, pages 1–17. Springer, 2013.

[26] Yehuda Lindell. Secure multiparty computation (mpc). *IACR Cryptol. ePrint Arch.*, 2020:300, 2020.

[27] Zachary C Lipton. The mythos of model interpretability: In machine learning, the concept of interpretability is both important and slippery. *Queue*, 16(3):31–57, 2018.

[28] Payman Mohassel and Peter Rindal. Aby 3: a mixed protocol framework for machine learning. In *Proceedings of the 2018 ACM Conference on Computer and Communications Security*, pages 35–52. ACM, 2018.

[29] Payman Mohassel and Yupeng Zhang. Secureml: A system for scalable privacy-preserving machine learning. In *2017 38th IEEE Symposium on Security and Privacy*, pages 19–38. IEEE, 2017.

[30] Moni Naor, Benny Pinkas, and Benny Pinkas. Efficient oblivious transfer protocols. In *Proceedings of the twelfth annual ACM-SIAM symposium on Discrete algorithms*, pages 448–457. Society for Industrial and Applied Mathematics, 2001.

[31] Valeria Nikolaenko, Udi Weinsberg, Stratis Ioannidis, Marc Joye, Dan Boneh, and Nina Taft. Privacy-preserving ridge regression on hundreds of millions of records. In *2013 IEEE Symposium on Security and Privacy*, pages 334–348. IEEE, 2013.

[32] Ziad Obermeyer, Brian Powers, Christine Vogeli, and Sendhil Mullainathan. Dissecting racial bias in an algorithm used to manage the health of populations. *Science*, 366(6464):447–453, 2019.

[33] Krzysztof Pietrzak. Cryptography from learning parity with noise. pages 99–114, 01 2012. ISBN 978-3-642-27659-0. doi: 10.1007/978-3-642-27660-6_9.

[34] RE Putra, AI Nurhidayat, and AY Wicaksono. Implementation of neural network to determine the new college students. In *IOP Conference Series: Materials Science and Engineering*, volume 288, page 012121. IOP Publishing, 2018.

[35] O.S Rothaus. On "bent" functions. *Journal of Combinatorial Theory, Series A*, 20(3): 300–305, 1976. ISSN 0097-3165. doi: https://doi.org/10.1016/0097-3165(76)90024-8. URL https://www.sciencedirect.com/science/article/pii/0097316576900248.

[36] Amartya Sanyal, Matt J. Kusner, Adrià Gascón, and Varun Kanade. TAPAS: tricks to accelerate (encrypted) prediction as a service. In *International Conference on Machine Learning*, pages 4497–4506, 2018.

[37] Shahar Segal, Yossi Adi, Benny Pinkas, Carsten Baum, Chaya Ganesh, and Joseph Keshet. Fairness in the eyes of the data: Certifying machine-learning models. *arXiv preprint arXiv:2009.01534*, 2020.

[38] Sameer Wagh, Divya Gupta, and Nishanth Chandran. Securenn: 3-party secure computation for neural network training. *Proceedings on Privacy Enhancing Technologies*, 1:24, 2019.

[39] Xiao Wang, Alex J. Malozemoff, and Jonathan Katz. EMP-toolkit: Efficient Multi-Party computation toolkit. https://github.com/emp-toolkit, 2016.

[40] Xiao Wang, Samuel Ranellucci, and Jonathan Katz. Authenticated garbling and efficient maliciously secure two-party computation. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 21–37. ACM, 2017.

[41] Xiao Wang, Samuel Ranellucci, and Jonathan Katz. Global-scale secure multiparty computation. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 39–56. ACM, 2017.

[42] Brian Hu Zhang, Blake Lemoine, and Margaret Mitchell. Mitigating unwanted biases with adversarial learning. In *Proceedings of the 2018 AAAI/ACM Conference on AI, Ethics, and Society*, pages 335–340. ACM, 2018.

A PROOFS FROM SECTION 4

Theorem 4.8. *Suppose C is given by Equation 1. If $\{h_k\}_{k\in K}$ is a family of random oracles then $\{C(h_k, \lambda)\}_{k\in K}$ is hiding.*

Proof. Let $r \leftarrow \mathcal{R}$. Suppose that a polynomial time algorithm A is given input k, $(H_k(i, r, x))_{i\in I}$. For fixed k, the $H_k(i, r, x)$ are independent uniform random variables irrespective of the value of x or r, so without querying the oracle the adversary can learn nothing about x or r.

When the adversary requests the value of the random oracle on an input beginning with $r' \in S$ suppose it is also told whether or not $r' = r$.

When the adversary queries with $r' \neq r$ it learns nothing about r except that $r \neq r'$. Thus the probability of using the right salt on the jth query is at most $1/(|\mathcal{R}| - j + 1)$ and so the probability of querying the correct r with a guesses is at most $a/|\mathcal{R}|$. As the adversary has time for only polynomially many queries and $|\mathcal{R}|$

No. of bits	Ours (MB)	SHA3-256 (MB)	Improvement
2^{14}	2.51	19.93	8×
2^{18}	10.90	300.34	28×
2^{22}	141.25	4805.39	34×
2^{26}	2169.02	76900*	35×
2^{30}	34022.77	1230200*	36×

Table 6: Comparison of communication for SHA3-256 baseline and our scheme for executing the assert **functionality. Here $p_c = 1/2$. * means estimated via extrapolation**

grows exponentially in λ it will query with r as the randomness with negligible probability.

Conditioned on A never querying the correct randomness, its view is independent of x and thus so is the probability of it outputting 1. □

B COMMUNICATION FOR assert.

Table 6 shows the amount of communication needed for executing the assert functionality using SHA3-256 and our scheme. We observe that our scheme requires 36× less communication for the committer and the verifier than the SHA3-256 based baseline.

C NUMBER OF INDICES $|\mathcal{I}|$

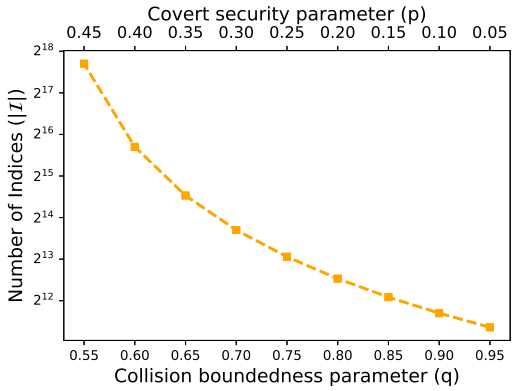

Figure 6: Number of indices (hashes) $|\mathcal{I}|$ needed to be computed by the committer as a function of q (and the covert security parameter p i.e. $1-q$). Here block size $b = 1024$ and statistical security parameter $\sigma = 40$.

We use the formulation, upon ceiling to the next nearest integer, defined in Theorem 4.13 to compute $|\mathcal{I}|$. In Figure 6, we plot this formulation for $\sigma = 40$, $b = 1024$ and different values of q (and the covert security parameter p i.e $1 - q$) to show how the number times $|\mathcal{I}|$ that the committer needs to compute H varies with the security parameters.

D CERTIFIED PREDICTIONS

In this section we describe how PVC committed MPC enables a key application, *certified predictions*: obtaining secure predictions by a private model that is certified to have certain properties (more on such properties below). We show by means of a real-world example how heuristic approaches that are sublinear in the input size fail. We do this by training a fair machine learning model and showing how, by modifying a single parameter of the model, it can be made unfair and more accurate. We describe previous work on the problem of obtaining predictions by a certified private model, and discuss an efficient solution enabled by our results.

	Accuracy			Average Odds Difference		
Dataset	Unfair	Fair	Changed	Unfair	Fair	Changed
Credit	69.3%	62.7%	64.3%	−0.341	0.359	0.122
COMPAS	67.6%	55.9%	65.0%	−0.181	0.369	0.073
Adult	80.4%	75.8%	78.8%	−0.270	0.261	0.111

Table 7: Accuracies of a fair prediction method [42] (Fair), the same model changed by a single weight to maximize accuracy (Changed), compared to the model trained without any constraints (Unfair).

Recently, ML models have started to be deployed into high-impact, real-world decision-making settings such as medicine [13], self-driving cars [10], and college admissions [34]. However, this has led to problems: many of these settings have key constraints that ML models were not originally designed to handle. Current models lack *interpretability* [27], *safety* [4], and *fairness* [32]. To address this, there has been a wealth of recent work aimed at formalizing these constraints and creating ML models that satisfy them [42**? ? ? ?**].

However, models that satisfy these constraints often have reduced accuracy as the constraints restrict the model's predictions in accuracy-agnostic ways. As model accuracy is often directly tied to beneficial outcomes (e.g., monetary investment, company profit, likelihood of publication), real-world constraints can incentivise service providers to cheat. To prevent this a natural question arises: *What is the minimal computation required to ensure cheating does not occur?* One may be tempted to try to construct a procedure that is sublinear in the size of the model. Recent work has proposed to generate a small series of tests to identify small changes to a model [20]. However, we show with a simple example that any protocol must ensure that nothing about the model is changed, requiring a linear time procedure.

Any change may sacrifice fairness. We investigate a popular real-world constraint placed on ML models: *fairness constraints*. In general, the most popular formulation of fairness constraints minimizes the difference between (functions of) predictions made on different demographic groups. Because these techniques constrain predictions across groups, their accuracy is less than unconstrained models. We investigate a popular fair prediction model [42] applied to three fair prediction problems: judging credit risk (Credit[2]); predicting parole violators (COMPAS[3]); inferring income (Adult[4]). We consider the following *average odds difference* fairness criterion

$$\left(\mathbb{E}[\hat{Y} \mid A=0, Y=y] - \mathbb{E}[\hat{Y} \mid A=1, Y=y]\right) \geq \tau, \quad \forall y \in \{0,1\},$$

where Y is the true outcome (e.g., $Y = 1$ signifies good credit in Credit, while $Y = 0$ signifies bad credit) and $\hat{Y}$ is the prediction. Here A indicates demographic group (e.g., race, gender, sexual orientation, among others). Specifically $A = 0$ indicates a *disadvantaged*

[2]https://tinyurl.com/cm-credit
[3]https://tinyurl.com/cm-compas
[4]https://tinyurl.com/cm-census

group and $A = 1$ indicates a *privileged group*. Thus the above constraint says that the average outcome for the disadvantaged group has to be at least τ-larger than the average outcome for the advantaged group. This is to combat predictors $\hat{Y}$ that benefit the privileged group (such predictors will arise from unconstrained training). These expectations are computed over a training dataset. Table 7 shows the accuracy and average odds difference of the model in Zhang et al. [42] using the fairness constraint (Fair), compared to the model without the fairness constraint (Unfair).

Now we imagine that a cheating service provider wants to take the fair model and only change a single element of the model to maximize accuracy. We imagine they test every single element, optimizing for accuracy alone, while fixing the remaining parameters. They then take the model which has the maximum improvement in accuracy across all single-parameter-changed models (Changed). We report the accuracy and fairness of this model in Table 7.

These results show that changing just a single element can significantly improve the accuracy over the fair model (by as much as 9.1% on COMPAS). Further, the changed model has significantly lower average odds difference than the fair model, unfairly benefiting the privileged group at the expense of the disadvantaged group. Thus, to ensure a service provider cannot surreptitiously improve accuracy at the expense of real-world constraints, a protocol must ensure that the entire model remains the same.

Related work. To prevent this a number of works have proposed techniques to verify ML models [17, 20, 24, 37]. SafetyNets [17] propose an interactive proof protocol for verifying deep neural network predictions. This protocol only has a verification guarantee and leaves a security guarantee to future work. Further it is limited to models expressible as arithmetic circuits. VerIDeep [20] describe a method to generate inputs for which small changes to the ML model would yield very different outputs. However, this model does not guarantee that the entire model remains the same and thus would be vulnerable to attacks similar to that described above.

Recent work with security guarantees [24, 37] propose to use hash functions (SHA-256, SHA-3 in sponge mode) to verify a model has not been altered. Specifically these works generate and verify a hash within MPC. In MPC the protocol cost is dominated by AND gate computations and the most efficient method requires asymptotically 35 AND gates per input bit [37]. While there exist an MPC-optimized hash called LowMCHash-256 [3] it is new and susceptible [14]. Our constructions above enable secure predictions with verified inputs that asymptotically require 0.5 AND gates per input bit and derives security from the well-known random oracle assumption.

Our approach. To enable certified predictions we propose the following procedure. First the service provider (committer, P1) makes a commitment $c = \mathsf{pvccommit}(x, \mathsf{sk}, r)$ to a model x. P1 then engages in an MPC protocol with a regulatory agency (P2') where P2' verifies the model x satisfies the required guarantee (e.g., fairness), and that c is a commitment to that model. If these checks pass then P2' signs the commitment c with their private key and sends it to P1. When a user (verifier, P2) wishes to obtain a certified prediction from P1, they engage in a PVC commitment. Here P1 sends c to P2. If (a) P2 can verify that c is signed by the regulatory agency P2' (e.g., this could be done if regulator's public key is publicly available) and (b) the PVC commitment is verified (via assert and check as described in Figure 1), then the output is a certified prediction.

E PROOFS AND DEFINITIONS OF SECTION 5 (INTEGRITY CHECKING)

E.1 Proof that we have constructed a PVC commitment scheme

Theorem 5.8. *If H is hiding and q-collision bounded and the signature scheme has deterministic verification and is discrimination resistant, then the above functions form a PVC commitment scheme with covert security parameter $p = 1 - q$ (Definition 5.1).*

Proof. To show that the functions pvccommit, assert and check for a PVC commitment scheme with parameter p, we must check that check is deterministic and that the four properties hold.

As the function check is given by a decision tree depending on checking whether (deterministic) parts of the input are equal and whether signatures are valid (which is deterministic by the assumption on the verification function) it is deterministic.

Correctness. Given i, r, x and a valid key pair sk, pk, let $c = \mathsf{pvccommit}(x, \mathsf{sk}, r)$ and $a = \mathsf{assert}(x, \mathsf{sk}, r; i, \mathsf{pk})$. Consider the definition of $\mathsf{check}(c, a, \mathsf{pk})$. As the key pair is valid both of the signatures will check out thus the result is not inconclusive. Furthermore, both $c[a[0]]$ and $a[1]$ are equal to $H(i, r, x)$, thus the check will return valid.

General Binding. Suppose, these functions do not satisfy general binding. Then there exists a polynomial time adversary, $\mathcal{A}$, contradicting Definition 5.4. Let $x, \mathsf{sk}, r, x', \mathsf{sk}', r', \mathsf{pk}$ and c be the output of this adversary. Further, let i, a, a', output and output$'$ be as in the definition. As the signature scheme is discrimination resistant the distribution of i conditioned on G (and thus on output or output$'$ being inconclusive) is still uniform. It follows that in order for Inequality 2 to hold we must have

$$\mathbb{P}(H(i, r, x) \neq c[i]) + \mathbb{P}(H(i, r', x') \neq c[i]) < p.$$

Thus for greater than a $1 - p$ fraction of the choices of i we must have $H(i, r, x) = c[i] = H(i, r', x')$. This would mean that r, x, r', x' is a q'-collision for some $q' > q$. The above process then gives a polynomial time algorithm contradicting the q-collision boundedness of H. So the general binding property must hold.

Hiding. Suppose $\mathcal{A}$ is a polynomial time adversary contradicting Definition 5.5. Consider the polynomial time algorithm that takes as input $(H(i, r, x))_{i \in \mathcal{I}}$, computes $\mathcal{A}(\mathcal{O}_{\mathsf{sk}}(x))$ (using a hard-coded sk) and outputs the result. This adversary contradicts the hiding property of H (Definition 4.5).

Defamation Freeness. Let $\mathcal{A}$ be a polynomial time adversary. In order to have $\mathsf{check}(c, a, \mathsf{pk}) = \mathsf{cheated}$ both c and a must be correctly signed. As the signature scheme is chosen-plaintext secure $\mathcal{A}$ can only achieve this with non-negligible probability by using the contents of $\mathcal{O}_{\mathsf{sk}}(x)$ as c and a (they ca not even be switched as they have different formats). But with that choice of c and a, $c[a[0]] = a[1]$, and thus the check would return valid. Therefore the functions are defamation free. □

E.2 Execution in the ideal world

Next, we present in detail the ideal world execution of a function $g(x, y)$. The ideal world is parameterized by the party corrupted by adversary $\mathcal{A}$, which we denote by $\mathsf{C} \in \{\mathsf{P1}, \mathsf{P2}, \perp\}$ ($\perp$ is just a value different from P1 and P2 to represent that all parties are honest) and, as mentioned above, two probabilities $p_{\mathsf{exec}}, p_{\mathsf{commit}}$. Let us remark that $\mathcal{A}$ has an auxiliary input, and that all parties are initialized with the same value on their security parameter tape (including the trusted party), but we leave both of these aspects implicit for clarity.

An unusual aspect of this ideal world is the presence of an "observation of the environment" which happens after the commitment has been made but before the computation. The idea being that a party has committed to an input if they are unable to make it depend on something they learnt between commitment and computation. We assume that this observation is drawn from some distribution $\mathcal{E}$ and that the distribution can be sampled from by a polynomial time algorithm. This latter assumption is to stop the environment from encoding, say, collisions of zero for a secure hash function.

1. P1 *receives input.* Party P1 receives its prescribed input x.

2. Commitment of P1*'s input.* At this stage P1 sends to the trusted party the input that it intends to use in a subsequent computation, which we denote by w. If $\mathsf{C} \neq \mathsf{P1}$, then $w = x$, and otherwise $\mathcal{A}$ sets w to be an arbitrary valid input value in a way that might depend on x.

3. The environment is revealed. The value e is sampled from the distribution $\mathcal{E}$ Party P1 is given the value of e

4. P2 *receives input and parties send inputs.* Party P2 receives its prescribed input y. Next, P1 and P2 send to the trusted party their inputs to be used in the computation, denoted a, b, respectively.

- If $\mathsf{C} = \mathsf{P1}$, then $\mathcal{A}$ sets $a \in \{w, \mathtt{abort}, \mathtt{corrupted}, \mathtt{cheat\_exec}, \mathtt{cheat\_commit}\}$ for P1, and otherwise P1 sets $a = w$.
- If $\mathsf{C} = \mathsf{P2}$ then $\mathcal{A}$ sets b for P2, otherwise P2 sets $b = y$.

5. Early abort & blatant cheating. $\mathcal{A}$ is given the opportunity to have C abort or announce that it is corrupted. This results in updating either a (if $\mathsf{C} = \mathsf{P1}$) or b (if $\mathsf{C} = \mathsf{P2}$) to abort or corrupted. If that is the case, a (resp. b) is forwarded to P2 (resp. P1) and the trusted party halts.

6. Attempted cheat option. If $a =$ cheat_exec, then the trusted party tosses a coin $X = \mathsf{Ber}(p_{\mathsf{exec}})$, where p_{exec} is the probability of P1 getting caught cheating at this stage, and

- If $X = 1$ then the trusted party sends corrupted to both P1 and P2.
- If $X = 0$ then the trusted party sends undetected to P1, along with y (P2's input). Following this, $\mathcal{A}$ gets to choose P2's output of the protocol, and sends it to the trusted party.

The ideal execution ends at this point if $a =$ cheat_exec.

7. Attempted break commitment option. If $a =$ cheat_commit then the trusted party requests from P1 (a) a probability q and (b) a new value w' for w. The trusted party then sets $p = q$, if $w = w'$, and $p = \max(q, p_{\mathsf{commit}})$ otherwise, where p_{commit} is the probability of P1 getting caught cheating at this stage. (Note that this simply allows the adversary to choose an arbitrary probability of getting caught when cheating to rewrite w with the same value again). Then, the trusted party (i) tosses a coin $Y = \mathsf{Ber}(p)$, (ii) rewrites w to take value w', (iii) runs $g_1(w, b)$ with the updated w, and (iv) gives $\mathcal{A}$ the opportunity to abort P1. Next,

- if $Y = 1$ then the trusted party sends corrupted to both P1 and P2 and halts, and
- if $Y = 0$ then the trusted party sends undetected to P1.

Let us remark that giving $\mathcal{A}$ the opportunity to abort upon observing the output in in step 7. is allowed just to simplify the presentation of out protocol, and that an extension where $\mathcal{A}$ does not receive an output when caught cheating is easy to achieve by just adding a round of interaction to our protocol. In that extra round P2 enables P1 to ungarble their output after verifying the commitment resulting from the secure computation.

8. Trusted party gives out outputs. The trusted party evaluates $g(w, b)$, and gives $\mathcal{A}$ the chance to abort the execution. Otherwise it gives their designated output to P2, at which point $\mathcal{A}$ is allowed to either abort the execution, or let the honest party receive their output.

Outputs. The honest party outputs what they received in the final step, and $\mathcal{A}$ outputs an arbitrary (probabilistic) polynomial-time computable function of C's input, any auxiliary input, and its view during the execution.

E.3 Definitions

The following simulation security definition deviates from most such definitions in that we allow the adversary in the ideal model to be logically omniscient, whereas it is standard to restrict the simulator to polynomial time computations. The polynomial time assumption is important in the context of zero-knowledge proofs and for certain systems of composability. However standard bit commitment is impossible in the universal composability model [11], thus we must settle for weaker composition guarantees here. We hope it is clear that the ideal setting here is information theoretically secure. Thus even a logically omniscient adversary can not possibly learn things that it should not in the ideal model. The simulator could be made computable at the expense of slightly complicating the proof, but as this is unnecessary and also not standard we prefer to keep the proof simple.

Denote by $\mathtt{IDEAL}^{p_{\mathsf{exec}}, p_{\mathsf{commit}}, \mathcal{E}}_{g, \mathcal{S}(z), i}(x, y, \lambda)$ the environment variable and the outputs of the honest parties and adversary in an execution in the ideal world above, and let $\mathtt{REAL}^{p_{\mathsf{exec}}, p_{\mathsf{commit}}, \mathcal{E}}_{g, \mathcal{A}(z), i}(x, y, \lambda)$ denote the environment variable and the outputs of the honest parties and the adversary in a real execution of a protocol π.

Definition E.1. Let g and p be as above. A protocol π *securely computes* g *with committed first input in the presence of a malicious P1 or a covert P2 with* p*-deterrent* if for every non-uniform probabilistic polynomial time adversary $\mathcal{A}$ for the real model, there exists a definable adversary $\mathcal{S}$ for the ideal model such that for each $i \in \{1, 2\}$:

$$\left\{\mathsf{IDEAL}^{p_{\mathsf{exec}},p_{\mathsf{commit}},\mathcal{E}}_{g,\mathcal{S}(z),i}(x,y,\lambda)\right\}_{x,y,z\in\{0,1\}^*,\lambda\in\mathbb{N}} \equiv^c \left\{\mathsf{REAL}^{p_{\mathsf{exec}},p_{\mathsf{commit}},\mathcal{E}}_{g,\mathcal{A}(z),i}(x,y,\lambda)\right\}_{x,y\in\{0,1\}^*,\lambda\in\mathbb{N}}$$

In PVC security it is important that a fail-stop adversary is not labelled a cheat (at least in most contexts including ours) for that we say that:

Definition E.2. A protocol π is *non-halting detection accurate* if for every fail-stop adversary $\mathcal{A}$ controlling party P1, the probability of an honest P2 outputting `corrupted` is negligible.

In order to have PVC security in place of the covert security we require that their be some algorithm `Blame`. When applied to the view of an honest party that has outputted `corrupted`, it must return a proof of that corruption. The proof is verified by another algorithm `Judgement`, which will output `cheated` if and only if it is a genuine proof. These ideas are formalized as follows.

Given an algorithm `Commit` and a protocol $\mathcal{P}$ let the commitment protocol formed by them consist of P1 running `Commit` and sending the result to P2, P1 then receiving $e \leftarrow \mathcal{E}$ and then both P1 and P2 engaging in $\mathcal{P}$ and taking the output from that protocol as the output.

Definition E.3. A quadruple (`Commit`, $\mathcal{P}$, `Blame`, `Judgement`) *securely computes g with committed first input in the presence of a malicious P1 or a covert P2 with p-deterrent and public verifiability* if the following hold:

(1) (Simulatability with p-deterrent:) The commitment protocol formed from `Commit` and $\mathcal{P}$ securely computes g with committed first input in the presence of a malicious P1 or a covert P2 with p-deterrent and is non-halting detection accurate.
(2) (Accountability:) For every PPT adversary $\mathcal{A}$ controlling P1 and interacting with an honest P2,

$$\mathbb{P}(\texttt{P2 outputs corrupted} \wedge \texttt{Judgement(Blame(View(P2)))} \neq \texttt{cheated})$$

is negligible.
(3) (Defamation-Free:) For every PPT adversary $\mathcal{A}$ controlling P2 and interacting with an honest P1,

$$\mathbb{P}(\mathcal{A}\ \text{outputs}\ \wedge \texttt{Judgement}(Cert) = \texttt{cheated})$$

is negligible.

E.4 PVC committed MPC Proof

Theorem 5.9. *The quadruple* (`Commit`, $\mathcal{P}$, `Blame`, `Judgement`) *securely computes g with committed first input in the presence of a malicious* P1 *or a covert* P2 *with $p/2$-deterrent and public verifiability.*

If pvccommit *and* check *are used as given in the previous section then we can replace the $\mathcal{P}$ with the protocol in Fig. 5 and still have the same security guarantee.*

Furthermore, if in either case it can be guaranteed that P1 *is honest in running the commitment algorithm in Fig. 3, then the deterrent factor improves from $p/2$ to p.*

Proof. **Simulating P1** First we consider simulatability in the case where P1 is corrupted. Given a non-uniform probabilistic polynomial time adversary $\mathcal{A}$, we construct $\mathcal{S}$ as follows.

First note that $\mathcal{S}$ can uniformly randomly choose a randomness tape which will be used for all of its black box runs of $\mathcal{A}$, this reduces the task to the case where $\mathcal{A}$ is a deterministic adversary.

By running $\mathcal{A}$ the simulator is given the commitment c that $\mathcal{A}$ chooses to use (which may or may not be generated by applying `Commit` to some w). Now $\mathcal{S}$ can look at how $\mathcal{A}$ would respond to every possible environment variable e (this is fine because it is a mathematical function which need not be computable). If, in response to e, $\mathcal{A}$ does any of early abort, blatant cheat or cheat during the execution the outcome is independent of what commitment was made so it would not matter what $\mathcal{S}$ commits to in the ideal world. The other possibility is that $\mathcal{A}$ does none of those things and submits some x' as their input alongside supposed randomness r'.

The underlying protocol Π allows input extraction in polynomial time (as is used in the proofs of security for that protocol in Hong et. al. [23]) thus in all of these other cases $\mathcal{S}$ can extract which x' will be used in response to each e. For each one $\mathcal{S}$ can then compute $a = \mathsf{assert}(x', \mathsf{sk}, r'; i, \mathsf{pk})$ for every $i \in \mathcal{I}$ and check for what fraction of the i we have $\mathsf{check}(c, a, \mathsf{pk}) = \mathsf{cheated}$.

For those e that result in being caught with probability at least $p/2$ it would not matter what $\mathcal{S}$ committed to as it will be able to attempt to cheat the commitment to change the input to x' and get caught with the correct probability. After which it will receive $g_1(x', y)$, add it to the simulated view, and proceed according to what $\mathcal{A}$ would do next. Aborting if and only if $\mathcal{A}$ chooses to abort. P2 will then receive `corrupted` with the correct probability.

Those e that result in less than $p/2$ probability of being caught, it will matter that $\mathcal{S}$ commited to the value of x' that $\mathcal{A}$ wants to use. Thus for the commitment phase $\mathcal{S}$ will commit with the trusted party to the value $\tilde{x}$ that is most likely to be used as x' (with respect to the randomness of e). If the adversary uses $x' = \tilde{x}$ then $\mathcal{S}$ will now be able to tell the trusted party it wants to use that value and it wants to get caught with the correct probability.

The remaining possibility, that e results in $\mathcal{A}$ using an input $x' \neq \tilde{x}$ and r' which has a probability less than $p/2$ of resulting in P1 being caught, would be a serious problem for $\mathcal{S}$. We claim however that this can happen with only negligible probability.

Suppose to the contrary that some non-negligible fraction of the weight of $\mathcal{E}$ resulted in these bad x', r'. Then as each must individually have weight at most that assigned to $\tilde{x}$ we can construct a polynomial time algorithm as follows.

Sample $e \leftarrow \mathcal{E}$ and extract the input of $\mathcal{A}$ for this e, compute $a = \mathsf{assert}(x', \mathsf{sk}, r'; i, \mathsf{pk})$ for all $i \in \mathcal{I}$, check to see if less than a $p/2$ fraction of the is would result in $\mathsf{check}(c, a, \mathsf{pk}) = \mathsf{cheated}$. Repeat this process until two distinct such values of (x', r') have been found with this property. As the fraction of e that result in finding such an x' other than the most common one is non-negligible, this algorithm runs in expected polynomial time.

However, this can (by putting a polynomial time upper bound on the run time and failing if it reaches it) be converted into a PPT algorithm which contradicts the general binding property of the PVC commitment scheme. Thus proving the claim.

This addresses simulating the correct distribution between e, the output of P2 and messages explicitly sent in our protocol to P1. The

messages sent to P1 in the secure computation black box are dealt with by the simulator for Π as given in Hong et. al. [23].

Extending this to the optimized integrity check case is straight forward, everything is the same except $\mathcal{S}$ must produce a fake hash-commitment for $\mathcal{A}$ to sign. This can be done by hashing randomness due to the hiding property of the commitment scheme this would not break indistinguishability. Further whilst the signed version should be given to P2 in the real world it does not form part of P2's output so we need not worry about coordinating with that.

If the commitment had been produced honestly, then $\mathcal{A}$ must have some x, r that it committed to that it knows of. This together with any value of x', r' that collides with the resulting commitment c less than some fraction p of the time will break general binding. Thus the same simulator as above with this extra observation gives the stronger security.

Simulating P2 Simulating the other side is much easier. P2 receives a commitment c to some x, however due to the hiding property of the commitment $\mathcal{S}$ can get away with providing $\mathcal{A}$ with a commitment to some arbitrary value, say 0. The simulator can now have $\mathcal{A}$ interact with a copy of P1 (multiple times) in order to extract their input y and index i. It can then give this input to the trusted party to find the correct value of $g_2(x, y)$. It can then add the result of $g_2(x, y)$ to the simulation of the adversaries view using the simulator for Π.

With the optimization the only change is that rather than giving the value of assert for the given i, a pair $(i, c[i])$ is signed and added to the simulated view.

Correctness If both parties are honest then Π will correctly output $(g_1, (x, y), g_2(x, y))$ and the result of assert. As assert is computed correctly on the honest inputs the check with the commitment from the first step will return valid. And thus P2 will accept the output and the parties will have successfully computed g.

Accountability For accountability, we need to show that a cheating P1 gets caught *publicly*, in the sense that if P2 claims that P1 cheated then there's a proof accompanying that claim except with negligible probability. Note that if P1 cheats inside the secure computation with Π, this follows from the PVC properties of Π, and the definition of Blame and Judgemenet in terms of $\mathtt{Blame}_{\mathtt{exec}}$ and $\mathtt{Judgement}_{\mathtt{exec}}$. If P1 cheats in that the input to Π differs from the committed value then $\mathtt{Judgement}_{\mathtt{commit}}(\mathtt{Blame}_{\mathtt{commit}}(\cdot))$ will output cheated after verifying that $\mathtt{Blame}_{\mathtt{commit}}(\cdot)$ constitutes a valid signature of the fact that the commitment c and evaluation of H at i do not match. This happen with all but negligible probability due to the properties of the cryptographic signature, and the correctness of the PVC commitment scheme, i.e. different values for indexed hashes necessarily come from different inputs.

Defamation Freeness Defamation freeness states that proofs of cheating can not be forged. This holds for proofs outputted by Π by the fact that it satisfies PVC security, and it holds for proofs generated by $\mathtt{Blame}_{\mathtt{exec}}$ due to the properties of the cryptographic signature scheme.

□

F PROOFS OF SECTION 7 (LOWER BOUNDS)

Proposition 7.1. *Given any non-trivially collision bounded family of indexed hash functions $\{H_k\}_{k\in K}$ with H_k given by the (polynomial size) circuit C_k with n-bit main input, and m-bit output. With all but negligible probability over the generation of $k = G(\lambda)$, the circuit C_k must have at least $\lceil (n-m)/2 \rceil$ nonlinear gates.*

Proof. We give a polynomial time algorithm which, given a circuit, C, that implements a function from $\{0,1\}^a \times \{0,1\}^n$ to $\{0,1\}^m$ and contains fewer than $\lceil (n-m)/2 \rceil$ nonlinear gates, returns a non-zero input $x \in \{0,1\}^n$ such that $C(s, x) = C(s, 0)$ for all $s \in \{0,1\}^a$. As this algorithm finds a 1-collision with certainty if the circuit is small enough, for H_k to be secure that must happen with negligible probability in λ. And the result is immediate.

Consider the wires of C that are either outputs of the circuit or inputs to nonlinear gates. The hypotheses imply that there are $< n$ of such wires. Wire j must contain the XOR of a linear (i.e. parity) function f_j of the input with an affine function of the key and nonlinear gate outputs (either of which could be trivial).

The conditions $f_j(x) = f_j(0)$ form a collection of $< n$ linear constraints in n variables. Since $x = 0$ is obviously a solution of this under-determined system, then it must have also a nontrivial solution, which can be found efficiently. □

Proposition 7.3. *Let $\{h_k\}_{k\in K}$ be a collision resistant family of hash functions with h_k given by the circuit C_k with n-bit input and m-bit output. With all but negligible probability with respect to the generation of $k = G(\lambda)$, the circuit C_k must have at least $n - m$ nonlinear gates.*

Proof. The idea of the proof is similar to that of Proposition 7.1. We give a polynomial time algorithm which given a circuit C from $\{0,1\}^n$ to $\{0,1\}^m$ with fewer than $n - m$ nonlinear gates returns a collision in that circuit. Thus to have collision resistance the circuit can be that small only with negligible probability.

Let d be the number of nonlinear gates in C, we will assume WLOG that they are AND gates. We will derive $d + m$ affine conditions on x which determine $C(x) = C(0)$. As affine systems are efficiently solved, collision resistance requires that there be at most one solution ($x = 0$) to this system. For this to happen $d + m \geq n$ must hold, which proves the statement.

Thus it remains to derive the aforementioned $d + m$ affine conditions. We need to fix the output of each AND gate using only one affine condition on x. This can be achieved as follows. Consider each AND gate in (a totalisation of the partial) order from input to output, i.e. a topological ordering of the circuit. For each AND in that sequence, if the first input can be set to 0 with an affine condition then add that condition to the set and move on to the next gate. Otherwise, the first input is already determined so we need only add an affine condition that fixes the second input. Either way that is only one condition per gate. In summary, by adding one condition for each of the output wires we can determine their values, so we are done. □

Proposition 7.4. *Suppose that $\{H_k\}_{k\in K}$ is a non-trivially collision bounded and hiding family of hash functions. Let H_k be given by C_k with an n-bit main input and d nonlinear gates, then with all but negligible probability, $d \geq n/5$. Further if $|\mathcal{I}| = 1$, then $d \geq n/3$.*

Proof. For a indexed hash function circuit C with n-bit main input and d nonlinear gates we explain how to do each of the following in time polynomial in the size of C:

- Transform C into another circuit $\tilde{C}$ with d non-linear gates.
- Simulate the output of $\tilde{C}(i,r,x)$, given the output of $C(i,r,x)$ and i.
- Derive a collision in $\tilde{C}$ from a collision in C

Finally we will show that $\tilde{C}_k$ has output length at most $3d_k$ with all but negligible probability. It follows that $\{\tilde{C}_k\}_{k\in K}$ is non-trivially collision bounded and the result follows from Propositions 7.1 and 7.3.

Throughout this proof L with a subscript will denote a linear function.

Let C be a circuit with input (i,r,x) where x is the n-bit input, $r \in \mathcal{R}$ is the randomness and i is an index. All the following computations can be done in polynomial time we will avoid repeating this fact for each one.

Note that $C(i,r,x)$ can be rewritten as $L_1(i,r,x,g(L_2(i,r,x)))$ for some nonlinear function g where L_2 has a $2d$ bit output and g has a d bit output and is implemented with d non-linear gates.

Considering L_2 as a linear function of $\mathcal{R}$ we can find its kernel T which has codimension at most $2d$. Compute representations of $\pi_{T^\perp}(r)$ and $\pi_T(r)$, represented in a basis of $T^\perp$ and a basis of T, call them r_1 and r_2 respectively. Thus r_1 has length at most $2d$, and $L_2(i,r,x)$ is equal to some $L_3(i,r_1,x)$. We can thus write $C(i,r,x)$ as

$$L_4(i) + L_5(r_2) + L_6(r_1, x, g(L_3(i,r_1,x)))$$

We now define $\tilde{C}(i,r_1,x)$ to be a representation of

$$\pi_{(\mathrm{Im}L_5)^\perp}(C(i,r,x) - L_4(i)) = L_7(x, r_1, g(L_3(i,r_1,x)))$$

in a basis of $\mathrm{Im}L_7$.

As $\tilde{C}(i,r,x)$ is a linear function of $C(i,r,x)$ and i we can write it with d nonlinear gates.

As $\tilde{C}(i,r_1,x)$ is a known linear function of i and $C(i,r,x)$ so simulating it is trivial.

We abuse notation and use r for the function that recovers r from a derived r_1 and r_2. Suppose that $\tilde{C}(i,r_1,x) = \tilde{C}(i,r_1',x')$ and $x \neq x'$. Then we can compute $C(i,r(r_1,0),x) - C(i,r(r_1',0),x')$ which by the definition of $\tilde{C}$ will be in $\mathrm{Im}L_5$, we can then choose r_2 such that

$$C(i,r(r_1,0),x) - C(i,r(r_1',0),x') = L_5(r_2)$$

which combined with Equation F yields

$$C(i,r(r_1,0),x) = C(i,r(r_1',r_2),x')$$

Finally, recall that $\tilde{C}(i,r_1,x)$ is a representation of $L_7(x,r_1,g(L_3(i,r_1,x)))$ and that $\tilde{C}$ has full rank. Thus we can write each $\tilde{C}_k$ as

$$L_8^k(x) + L_9^k(r_1, g(L_3(i,r_1,x)))$$

As $(r_1, g(L_3(i,r_1,x)))$ is at most $3d$ bits long the rank of L_9^k must be at most $3d$. If $\mathrm{Im}L_8^k$ is contained in $\mathrm{Im}L_9^k$ then the length of the output of $\tilde{C}_k$ is at most $3d$. Otherwise, $\pi_{(\mathrm{Im}L_9^k)^\perp}\tilde{C}_k(i,r,x)$ is a non-trivial linear function of x, however this latter possibility must occur with negligible probability otherwise $\{\tilde{C}_k\}_{k\in K}$, and thus $\{C_k\}_{k\in K}$, is not hiding. □

G PROOFS OF SECTION 8 (FROM COVERT TO MALICIOUS)

Theorem 8.1. *If $\{H_k\}_{k\in K}$ is q-collision bounded then $\{H_k^E\}_{k\in K}$ is q^κ-collision bounded.*

Furthermore, the number of AND and XOR gates required to compute H_E is $l\lceil n/w\rceil$ times the number of gates required per bit by H plus $\lceil n/w\rceil$ times the number of gates required by E.

Proof. Suppose x and x' are two distinct inputs. Then for some j, $x^j \neq x'^j$. Then by the error detecting property $E(x^j)$ and $E(x'^j)$ must differ in at least κ places, $j_1, ..., j_\kappa$. Thus $\tilde{x}_{j_m} \neq \tilde{x}'_{j_m}$ for all $m \in \{1, ..., \kappa\}$.

Suppose that r, x, r', x' is a q'^κ-collision of H_k^E for some $q' > q$. Then at least one of $r, \tilde{x}_{j_m}, r', \tilde{x}'_{j_m}$ is a q'-collision of H_k. Thus, by testing each m in turn, a q'-collision of H_k can be found in polynomial time from a q'^κ-collision of H_k^E. The first part of the result follows. The number of required gates follows from counting through the algorithm for H^E. □

H PROOFS OF SECTION 9 (ARITHMETIC CIRCUITS)

Theorem 9.1. *Construction 4 is $b/|\mathcal{I}|$-collision bounded.*

Proof. By Theorem 4.9 it suffices to show that for two distinct inputs x, x' at most b values of i will result in $d_4(i,x) = d_4(i,x')$. Consider two distinct inputs x, x', and assume WLOG that they differ in the first b field elements. Let y and y' be the first b field elements from x and x' respectively.

It suffices to show that only b values of i will result in $d_4(i,y) = d_4(i,y')$. Start by expanding out the difference.

$$d_4(i,y) - d_4(i,y') = \sum_{j=1}^{b/2} (i^{2j-1}(y_{2j} - y'_{2j}) + i^{2j}(y_{2j-1} - y'_{2j-1}) + y_{2j-1}y_{2j} - y'_{2j-1}y'_{2j})$$

Let s be the function on the natural numbers that swaps $2j$ and $2j-1$ for all j. The difference is given by the inner product of $(1, i, i^2, ..., i^b)$ with a vector $v(y,y')$ in $\mathbb{F}^{b+1}$ with zeroth entry being

$$\sum_{j=1}^{b/2} y_{2j-1}y_{2j} - y'_{2j-1}y'_{2j}$$

and jth entry for $j > 0$ being $y_{s(j)} - y'_{s(j)}$.

We have $d_4(i,y) = d_4(i,y')$ only if $(1, i, ..., i^b)$ is perpendicular to $v(y,y')$. But as any $b+1$ vectors of the form $(1, i, ..., i^b)$ form a Vandermonde matrix and thus are linearly independent at most b of them could be perpendicular to any fixed $v(y,y')$. □