Independent Zero-Knowledge Sets

Rosario Gennaro* and Silvio Micali**

Abstract. We define and construct *Independent Zero-Knowledge Sets (ZKS) protocols*. In a ZKS protocols, a Prover commits to a set S, and for any x, proves non-interactively to a Verifier if $x \in S$ or $x \notin S$ without revealing any other information about S. In the *independent* ZKS protocols we introduce, the adversary is prevented from successfully correlate her set to the one of a honest prover. Our notion of independence in particular implies that the resulting ZKS protocol is non-malleable.
On the way to this result we define the notion of *independence* for commitment schemes. It is shown that this notion implies non-malleability, and we argue that this new notion has the potential to simplify the design and security proof of non-malleable commitment schemes.
Efficient implementations of ZKS protocols are based on the notion of *mercurial commitments*. Our efficient constructions of independent ZKS protocols requires the design of *new* commitment schemes that are simultaneously independent (and thus non-malleable) and mercurial.

1 Introduction

The notion of Zero Knowledge Sets (ZKS) was recently introduced by Micali, Rabin and Kilian in [19]. In these protocols, one party (Alice) holds a secret database Db which can however be accessed by another party (Bob) via queries. When Bob queries the database on a key x, Alice wants to make sure that nothing apart from $Db(x)$ is revealed to Bob, who at the same time wants some guarantee that Alice is really revealing the correct value.

Micali *et al.* presented a very ingenious solution to this problem, based on a new form of commitment scheme (later termed *mercurial commitments* in [16]). In a nutshell, Alice first commits to the entire database in a very succinct way, and then when Bob queries a given key x, Alice answers with a "proof" π_x that $Db(x) = y$ according to the original commitment. Their solution is efficient and based on the discrete logarithm assumption.

A construction based on general assumptions, and allowing more general queries on the database, was presented in [21]. However their construction required generic ZK proofs, based on Cook-Levin reductions and thus was less efficient than [19]. The original construction in [19] has been generalized to hold under various assumptions in [16] and [5].

Malleability. ZKS protocols guarantee simply that when Bob queries x, only the value of $D(x)$ is disclosed. However, this is only one of possible attacks

* IBM Research. `rosario@watson.ibm.com`
** MIT CSAIL.

that can be carried on a cryptographic protocol. It is well known that proving confidentiality may not be sufficient, in an open network like the Internet, where an Adversary can play the role of "man-in-the-middle" between honest parties.

First formalized in [11], the notion of malleability for cryptographic protocols describes a class of attacks in which the adversary is able to correlate her values to secret values held by honest players. In a ZKS protocol, for example, this would take the form of the adversary committing to a set somewhat related to the one of a honest player and then using this to her advantage.

The confidentiality property of ZKS protocols does not prevent such an attack from potentially taking place. Indeed such an attack could be devised against the protocol from [19]. What we need is an enhanced definition of security, to make sure that databases committed by one party are independent from databases committed to by a different party.

Non-Malleable Commitments. The first non-malleable commitment scheme was presented in [11], but it required several rounds of communication. A breaktrough result came with a paper by Di Crescenzo, Ishai and Ostrovsky (DIO) [9] which constructed a non-interactive and non-malleable commitment scheme. Following the DIO approach several other commitment schemes were presented with improved efficiency or security properties (e.g. [10, 8, 17, 13]).

The DIO approach has a very interesting feature: non-malleability is proven by showing that the commitment satisfies a basic "independence" property (though this property in not formally defined as such), and then it is shown that this property implies non-malleability. All the commitment schemes that followed the DIO approach have a proof of security structured in a similar way. However the only "original" part of the proof in each scheme is the proof that the commitment satisfies this "independence" property. The second part of the proof is basically identical in all the proofs.

Our Contribution.

- We define the notion of *Independent Zero Knowledge Sets* which enforces the independence of databases committed by various parties. We also define the notion of independence for *commitment schemes*. This definition captures the crucial notion of security in a DIO-like commitment.

  Once this notion of independence is formalized we restate the second part of the DIO proof as a formal theorem that shows once and for all that independent commitments are non-malleable.

  We believe that isolating the notion of independence has the potential to simplify the design and security proof of non-malleable commitments in the future.
- We present efficient independent ZKS protocols. These protocols are enabled by the efficent constructions of new commitment schemes that are simultaneously independent (and thus non-malleable) and mercurial.
- Finally we define various notions of non-malleability for ZKS protocols. We then ask if the DIO theorem (that independence implies non-malleability for commitments) holds for ZKS protocols as well. Surprisingly the answer is not

that simple. We show under which conditions independent ZKS protocols are also non-malleable.

PRELIMINARIES. In the following we are going to need several cryptographic tools and assumptions, with which we assume the reader is familiar. They are however recalled in Appendix A.

2 Zero-Knowledge Sets

ZKS DEFINITION. An *elementary database* Db is a subset of $\{0,1\}^* \times \{0,1\}^*$ such that if $(x,v) \in Db$ and $(x,v') \in Db$ then $v = v'$. With $[Db]$ we denote the *support* of Db, i.e. the set of $x \in \{0,1\}^*$ for which $\exists v$ such that $(x,v) \in Db$. We denote such unique v as $Db(x)$; if $x \notin [Db]$ then we also write $Db(x) = \perp$. Thus Db can be thought of as a partial function from $\{0,1\}^*$ into $\{0,1\}^*$.

In a ZKS protocol we have a Prover and Verifier: the Prover has as input a secret database Db. The Prover runs in time polynomial in $|[Db]|$ (the cardinality of the support) and the number of queries, while the Verifier runs in time polynomial in the maximal length of $x \in [Db]$, which we assume to be publicly known. They also have a common input string σ, which can be a random string (in which case we say that we are in the *common random string* model) or a string with some pre-specified structure (in which case we say we are in the *common parameters* model).

The Prover first commits in some way to the database Db. This commitment string is then given as input to the Verifier. Then the Verifier asks a query x and the Prover replies with a string π_x which is a proof about the status of x in Db. The Verifier after receiving π_x outputs a value y (which could be $\perp$) which represents his belief that $Db(x) = y$, or *bad* which represents his belief that the Prover is cheating.

A ZK Set protocol must satisfy completeness, soundness and zero-knowledge. Informally completeness means that if $Db(x) = y$ the Prover should always be able to convince the verifier of this fact. Soundness means that no efficient Prover should be able to produce a commitment to Db, a value x and two proofs π_x, π'_x that convince the Verifier of two distinct values for $Db(x)$. Finally zero-knowledge means that an efficient verifier learns only the values $Db(x)$ from his interaction with the Prover, and nothing else. In particular the Verifier does not learn the values $Db(x')$ for an x' not queried to the Prover (following [14] this is stated using a simulation condition). A formal definition appears in Appendix B.

2.1 Mercurial Commitments

A mercurial commitment scheme [16] is a commitment with two extra properties:

1. On input a message m, the sender can create two kinds of commitments: a *hard* and a *soft* commitment.
2. There are two kinds of openings: a regular opening and a partial opening or *teasing*.

The crucial properties of a mercurial commitment are: (i) both hard and soft commitments preserve the secrecy of the committed message (semantic security); (ii) hard commitments are indistinguishable from soft ones; (iii) soft commitments cannot be opened, but can be teased to any value (even without knowing any trapdoor information); (iv) hard commitments can be opened or teased only to a single value (unless a trapdoor is known). A formal definition from [5] appears in Appendix C.

A construction of mercurial commitments was implicitly presented in [19] based on discrete log. More constructions were presented in [16, 5], including one based on general assumptions. Let us recall the discrete log construction and a new one based on RSA[1].

Mercurial Commitments based on Discrete Log. This commitment is based on [1, 20]; the mercurial property was introduced in [19]. The public information is a cyclic group G of prime order q, where multiplication is easy and the discrete log is assumed to be hard. Also two generators g, h for G.

To hard commit to M, choose $\rho, R \in_r Z_q$: let $h_\rho = g^\rho h$ and commit using Ped-Com with bases g, h_ρ i.e. compute $C = g^M h_\rho^R$. The hard commitment is h_ρ, C. The opening is M, R, ρ and the verification of a hard commitment is to check the above equations. To soft commit, choose $\rho, R \in_r Z_q$: let $h_\rho = g^\rho$ and commit to 0 using Ped-Com with bases g, h_ρ i.e. compute $C = h_\rho^R$. The soft commitment is h_ρ, C Notice that in a soft commitment, one actually knows the discrete log of h_ρ with respect to g, while in a hard-commitment computing such discrete log is equivalent to computing $\log_g h$. Thus to tease the above soft commitment to M', one produces R' with $R' = R - M'\rho^{-1} \bmod q$. The verification of such a teasing consists in checking that $g^{M'} h_\rho^{R'} = C$.

Mercurial Commitments based on RSA. This commitment is based on [6, 7]; the mercurial property is an original contribution of this paper. The public information is an RSA modulus N, a prime e, such that $GCD(e, \phi(N)) = 1$; and $s \in_R Z_N^*$. To hard commit to M, choose $\rho, R \in_R Z_N^*$: let $s_\rho = s\rho^e \bmod N$ and commit using RSA-Com with base s_ρ i.e. compute $C = s_\rho^M R^e$. The hard commitment is s_ρ, C. The opening is M, R, ρ and the verification of a hard commitment is to check the above equations. To soft commit, choose $\rho, R \in_r Z_N^*$: let $s_\rho = \rho^e$ and commit to 0 using RSA-Com with base s_ρ i.e. compute $C = R^e$. The soft commitment is s_ρ, C Notice that in a soft commitment, one actually knows e-root of s_ρ, while in a hard-commitment computing such root is equivalent to computing the e-root of s. Thus to tease the above soft commitment to M', one produces R' with $R' = R\rho^{-M'} \bmod N$. The verification of such a teasing consists in checking that $s_\rho^{M'}(R')^e = C \bmod N$.

2.2 Constructing ZK Sets

Using any mercurial commitment it is possible to construct a ZKS protocol as shown in [19].

---

[1] This construction of mercurial commitments based on RSA was independently discovered in [16].

Let l to denote the maximal length of an input $x \in [Db]$. As we said above we assume this to be a publicly known value. The Prover uses a variation of a Merkle tree [18]. The Prover builds a tree of height l and stores a commitment to $Db(x)$ in the x-leaf (notice that if $x \notin [Db]$ then $Db(x) = \perp$). Then the Prover stores in each internal node a commitment to the contents of its two children: this is done by hashing the values of the two children using a collision-resistant hash function and then committing to the resulting value. The final commitment to Db is the value stored at the root. To prove the value of $Db(x)$ the Prover just decommits all the nodes in the path from the root to x (in particular this means that he reveals the values stored at their siblings, but without decommitting them), thus providing a Merkle-authentication path from the leaf to the root. The Verifier checks this path by checking the all the decommitments are correct.

Unfortunately the above algorithm runs in time 2^l, no matter what the size of the database is. In order to have the Prover run in time polynomial in $|[Db]|$, a pruning step is implemented as follows. First of all, we use mercurial commitments, to compute the commitments. In the above tree, we consider all the maximal subtrees whose leaves are *not* in $[Db]$. We store a soft-commitment in the roots of those trees. The rest of the tree is computed as above, using the hard commitments. Now the running time of the Prover is at most $2l|[Db]|$ since it is only computing full authentication paths for the leaves inside Db.

The question is now how do you prove the value of $Db(x)$. If $x \in [Db]$ then you just decommit (open) the whole authentication path from its leaf to the root, as before.

Let x be the a query such that $x \notin [Db]$, i.e. $Db(x) = \perp$. Let y be the last node on the path from the root to x that has a commitment stored in it[2]. We associate soft-commitments to the nodes on the path from y to x and their siblings, including x. Then we compute an authentication path from the root to x, except that we *tease* (rather than open) each commitment to the hash of the commitment of the children. Notice that we can seamlessly do this from the root to x. Indeed from the root to y these are either hard or soft commitments, and we only tease the hard ones to their real opening. From y to the leaf those are soft commitments that can be teased to anything.

A full detailed description of the protocol and its proof of security from [19, 16] is described in Appendix D.

3 Independent Zero-Knowledge Sets

Independent Commitments. As we said in the introduction, our starting point was the DIO approach [9] to build non-malleable commitments. In order to prove the non-malleability of their commitment scheme they first proved the following property.

[2] If x is the first query such that $Db(x) = \perp$, then y is the root of the maximal subtree that contains x, and such all its leaves are not in $[Db]$. However, as the Verifier asks queries $x \notin [Db]$ we start "filling up" that subtree with commitments and so we may need to go deeper in the tree to find the last commitment stored

Consider the following scenario: ℓ honest parties[3] commit to some messages and the adversary, after seeing their commitment strings, will also produce a commitment value. We require that this string must be different from the commitments of the honest parties (otherwise the adversary can always copy the behavior of a honest party and output an identical committed value). At this point the value committed by the adversary is *fixed*, i.e. no matter how the honest parties open their commitments the adversary will always open in a unique way.

In [9] this property is not formally defined but it is used in a crucial way in the proof of non-malleability. We put forward a formal definition for it (presented in Appendix E) and we say that such a commitment scheme is *ℓ-independent*. If it is ℓ-independent for any ℓ (polynomial in the security parameter) we say that is simply *independent*.

As mentioned in the Introduction, following the DIO approach, several other non-malleable commitments were presented (e.g. [10, 8, 17, 13]). *All* these schemes are independent according to our definition. Moreover their non-malleability proofs all share the same basic structure: an "original" part which proves that they are independent (once one formalizes the notion, as we did) – and a second part (common to all proofs and which basically goes back to DIO [9]) that the independence property implies non-malleability.

By formalizing the notion of independence we can then rephrase this second part of the DIO proof as an independent theorem (see Appendix F where the notion of ϵ-non-malleability for commitments is also recalled).

Theorem 1 (DIO [9]). *If an equivocable commitment scheme is ℓ-independent then it is (ℓ, ϵ)-non-malleable with respect to opening, for any ϵ. As a consequence, if an equivocable commitment scheme is independent then it is ϵ-non-malleable with respect to opening, for any ϵ.*

3.1 Defining Independence for ZK Sets

Let us consider a man-in-the-middle attack for a ZK Sets protocol. In such an attack, the Adversary would interact with the Verifier but while on the background is acting as a verifier himself with a real Prover. Of course we can't prevent the Adversary from relaying messages unchanged from the Prover to the Verifier and vice versa. But we would like to prevent an adversary to commit to a related database to the one committed by the real Prover and then manage to convince the Verifier that $Db(x)$ is a value different than the real one. When we define *independence* for ZKS our goal will be to prevent this type of attacks.

A WEAK DEFINITION. A first approach is to treat ZKS protocols in a similar way as commitments. Then the definition of independence would go as follows. The

[3] To be precise in [9] only the case $\ell = 1$ is considered, and suggested how to easily extend it to constant ℓ. The case of arbitrary ℓ (polynomial in the security parameter) is presented by Damgård and Groth in [8] to construct *reusable* non-malleable commitments.

adversary commits to a set after seeing the commitment of the honest prover, but *before* making any queries about the set committed by the honest prover. What we would like at this point is that the set committed by the adversary is *fixed*, i.e. it will not depend on the answers that the honest prover will provide on queries on his own committed set.

We call the above *weak independence*. This property is easily achieved by combining any ZKS protocol with any independent commitment. A formalization of this definition, the details and proof of this construction are in Appendix G.

A Stronger Definition. It may not be reasonable to assume that the adversary does not query the honest provers before committing. Thus a stronger definition of independence allows such queries. However once the adversary has seen the value $Db(x)$ of x in the database Db held by the honest prover, it can always commit to a set which is related to Db by the mere fact that the adversary *knows* something about Db (for example the adversary could committ to Db' where $Db'(x) = Db(x)$).

The idea is to make sure that the set committed by the adversary is *independent from the part of the honest prover's set that the adversary has not yet seen.* Here is how we are going to formalize this.

Consider an adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ which tries to correlate its database to the one of a honest prover. $\mathcal{A}_1$ sees the commitments of the honest provers, queries them (concurrently) on some values, and then outputs a commitment of its own. $\mathcal{A}_2$ is given concurrent access to the provers to whom he can ask several database queries while answering queries related to $\mathcal{A}_1$'s commitment. We would like these answers to be independent from the answers of the honest provers *except the ones provided to* $\mathcal{A}_1$ *before committing.*

In other words $\mathcal{A}_1$, after seeing the commitments $Com_1, \ldots, Com_\ell$ of ℓ honest provers, does the following

- queries Com_i on some set Q_i of indices, which are answered according to some database Db_i, and then
- outputs a commitment Com of its own.

We now run two copies of $\mathcal{A}_2$, in the first we give him access to provers that "open" the Com_i according to the databases Db_i; in the second instead we use some different databases Db'_i. However the restriction is that Db'_i *must agree with* Db_i *on the set of indices* Q_i. At the end $\mathcal{A}_2$ outputs a value x and the corresponding proof π_x with respect to Com. We require that the database value associated to x in the two different copies of $\mathcal{A}_2$ must be the same, which implies that it is "fixed" at the end of the committing stage.

Of course we must rule out the adversary that simply copies the honest provers, as in that case achieving independence is not possible (or meaningful). Thus we require that $\mathcal{A}_1$ output a commitment Com different from the honest provers' Com_i. A formal definition follows.

Given two databases Db, Db' and a set of indices Q we define the operator $\dashv$ as follows: $Db' \dashv_Q Db$ is the database that agrees with Db' on all the indices except the ones in Q where it agrees with Db.

We say that a ZKS protocol is *ℓ-independent* if the following property holds (where Q_i is the list of queries that $\mathcal{A}_1$ makes to the oracle $\mathsf{Sim2}^{Db_i(\cdot)}(\omega_i, Com_i)$):

ZKS ℓ-independence For any adversary $(\mathcal{A}_1, \mathcal{A}_2)$ and for any pair of ℓ-tuple of databases $Db_1, \ldots, Db_\ell$ and $Db'_1, \ldots, Db'_\ell$ the following probability

$$Pr\left[\begin{array}{c} (\sigma, \omega_0) \leftarrow \mathsf{Sim0}(1^k)\ ;\ (Com_i, \omega_i) \leftarrow Sim1(\omega_0)\ \forall\ i = 1, \ldots, \ell\ ; \\ (Com, \omega) \leftarrow \mathcal{A}_1^{\mathsf{Sim2}^{Db_i(\cdot)}(\omega_i, Com_i)}(\sigma, \omega)\ \text{ with }\ Com \neq Com_i\ \forall i\ ; \\ (x, \pi_x) \leftarrow \mathcal{A}_2^{\mathsf{Sim2}^{Db_i(\cdot)}(\omega_i, Com_i)}(\sigma, \omega)\ ; \\ (x, \pi'_x) \leftarrow \mathcal{A}_2^{\mathsf{Sim2}^{Db'_i \dashv_{Q_i} Db_i(\cdot)}(\omega_i, Com_i)}(\sigma, \omega)\ : \\ bad \neq \mathsf{V}(\sigma, Com, x, \pi_x) \neq \mathsf{V}(\sigma, Com, x, \pi'_x) \neq bad \end{array}\right]$$

is negligible in k.

The above notion guarantees independence only if the adversary interacts with a bounded (ℓ) number of honest provers. We say that a ZKS protocol is *independent* if it is ℓ-independent for any ℓ (polynomial in the security parameter). In this case independence is guaranteed in a fully concurrent scenario where the adversary can interact with as many honest parties as she wants.

THE STRONGEST POSSIBLE DEFINITION. A stronger definition allows $\mathcal{A}_1$ to copy one of the honest provers' commitments, but then restricts $\mathcal{A}_2$ somehow. Namely, we say that either $Com \neq Com_i$ for all i, or if $Com = Com_i$ for some i, the answer of $\mathcal{A}_2$ must be "fixed" on all the values x which she does not ask to the i^{th} prover. We call this *strong independence*.

We say that a ZKS protocol is *strongly ℓ-independent* if the following property holds, where Q_i (resp. Q'_i) is the list of queries that $\mathcal{A}_1$ (resp. $\mathcal{A}_2$) makes to the oracle $\mathsf{Sim2}^{Db_i(\cdot)}(\omega_i, Com_i)$:

ZKS strong ℓ-independence For any adversary $(\mathcal{A}_1, \mathcal{A}_2)$ and for any pair of ℓ-tuple of databases $Db_1, \ldots, Db_\ell$ and $Db'_1, \ldots, Db'_\ell$ the following probability

$$Pr\left[\begin{array}{c} (\sigma, \omega_0) \leftarrow \mathsf{Sim0}(1^k)\ ;\ (Com_i, \omega_i) \leftarrow Sim1(\omega_0)\ \forall\ i = 1, \ldots, \ell\ ; \\ (Com, \omega) \leftarrow \mathcal{A}_1^{\mathsf{Sim2}^{Db_i(\cdot)}(\omega_i, Com_i)}(\sigma, \omega)\ ; \\ (x, \pi_x) \leftarrow \mathcal{A}_2^{\mathsf{Sim2}^{Db_i(\cdot)}(\omega_i, Com_i)}(\sigma, \omega)\ ; \\ (x, \pi'_x) \leftarrow \mathcal{A}_2^{\mathsf{Sim2}^{Db'_i \dashv_{Q_i} Db_i(\cdot)}(\omega_i, Com_i)}(\sigma, \omega)\ : \\ bad \neq \mathsf{V}(\sigma, Com, x, \pi_x) \neq \mathsf{V}(\sigma, Com, x, \pi'_x) \neq bad \text{ AND} \\ ((Com \neq Com_i\ \forall i) \text{ OR } (\exists i : Com = Com_i \text{ AND } x \notin Q'_i)) \end{array}\right]$$

is negligible in k.

Again we say that a ZKS protocol is (strongly) independent if it is (strongly) ℓ-independent for any ℓ (polynomial in the security parameter).

Remark: Notice that because we need to open the same database commitment according to two different databases, the definition is stated in terms of the simulated provers. But since simulated executions are indistinguishable from real ones, we get that the independence property holds in real life too, in the sense that no matter how the honest provers open their committed database values, the adversary's openings are *fixed*.

This is the reason why we restrict the database Db'_i to agree with Db_i on the queries that were asked by the adversary before committing. In our proofs of security this requirement does not matter[4]. But the simulated execution is indistinguishable from the real one only if the answers are consistent. Thus after $\mathcal{A}_1$ has seen a given value for $Db_i(x)$, we need to make sure that in both copies of $\mathcal{A}_2$ the same value appears for $Db_i(x)$, in order to make the simulated run indistinguishable from a real one.

3.2 Constructing Independent ZKS

In this section we show how to modify the original protocol presented in [19] (recalled in Section 2.2) using a different type of commitment which will yield strong independence. This new commitment schemes that we introduce are simultaneously independent and mercurial.

Strong 1-Independence Based on Discrete Log The starting point of this protocol is Pedersen's commitment, modified as in [19] to make it mercurial. In order to achieve independence we modify the commitment further, using techniques inspired by the non-malleable scheme in [10]. We are going to describe the protocol that achieves strong independence and later show how to modify it if one is interested in just independence.

DLSI-ZKS

- *CRS Generation.* On input 1^k selects a cyclic group G of order q, a k-bit prime, where the discrete logarithm is assumed to be intractable and multiplication is easy. It also chooses three elements $g_1, g_2, h \in_R G$. Finally it selects a collision-resistant function H with output in Z_q. The CRS is $\sigma = (G, q, g_1, g_2, h, H)$
- *Prover's Committing Step.* On input Db and the CRS σ. Choose a key pair sk,vk for a signature scheme. Let $\alpha = H(\mathsf{vk})$ and $g_\alpha = g_1^\alpha g_2$. Run the prover's committing step from [19] on Db and the mercurial commitment defined by $\sigma_\alpha = (G, q, g_\alpha, h)$ to obtain Com, Dec. Output Com, vk.
- *Prover's Proving Step.* On input x compute π_x with respect to σ_α, Com, Dec using the prover's proving step from [19]. Then output Com, π_x and sig_x a signature on (Com, x) using sk.
- *Verifier.* Check that sig_x is a valid signature of (Com, x) under vk; if yes, compute $\alpha = H(\mathsf{vk})$ and $g_\alpha = g_1^\alpha g_2$ and run the [19] Verifier on $(\sigma_\alpha, Com, x, \pi_x)$, otherwise output *bad*.

[4] I.e. the adversary would not be able to output (x, π_x, π'_x) such that V outputs different values for $Db(x)$ depending on which proof, π_x or π'_x, is provided, *even if* Db' does not agree with Db_i on the set Q_i.

Theorem 2. *Under the discrete logarithm assumption,* DLSI-ZKS *is a strong 1-independent zero-knowledge set protocol.*

Proof appears in Appendix H. Also ℓ-independence can be obtained by standard replication techniques.

Strong Independence under the Strong RSA Assumption In this section we are going to use the mercurial RSA commitment described in Section 2.1. In order to achieve independence we are going to modify it further using techniques inspired by [8, 13] and use it inside MRK-ZKS. The modifications required to prove independence require the Strong RSA assumption. Here is a description of the protocol.

SRSA-ZKS

- *CRS Generation.* The key generation algorithm chooses a k-bit modulus, N as the product of two large primes p, q and a random element $s \in_R Z_N^*$. Also selects a collision-resistant hash function H which outputs prime numbers[5] $> 2^{k/2}$. Notice that such primes are relatively prime to $\phi(N)$. The CRS is $\sigma = (N, s, H)$
- *Prover's Committing Step.* On input Db and the CRS σ. Choose a key pair sk,vk for a signature scheme. Let $e = H(\mathsf{vk})$. Run the Prover's committing step from [19] on Db and the mercurial commitment defined by $\sigma_e = (N, s, e)$ to obtain Com, Dec. Output Com, vk.
- *Prover's Proving Step.* On input x compute π_x with respect to σ_e, Com, Dec using the Prover's proving step from [19]. Then output Com, π_x and sig_x a signature on (Com, x) using sk.
- *Verifier.* Check that sig_x is a valid signature of (Com, x) under vk; if yes, compute $e = H(\mathsf{vk})$ and run the [19] Verifier on $(\sigma_e, Com, x, \pi_x)$, otherwise output *bad*.

Theorem 3. *Under the Strong RSA Assumption,* SRSA-ZKS *is a strong independent zero-knowledge set protocol.*

Proof appears in Appendix I.

If one is interested in simple independence (rather than strong independence) both of the above protocols can be simplified by using more efficient one-time signature schemes for vk and just sign Com. It is easy to verify that the proof goes through. Even more efficiently, to obtain independence, one can use a message authentication code in place of a signature scheme (the original idea in [10]). Informally, the basic idea is to commit to a random MAC key a using a basic trapdoor commitment: call this commitment A. Set $\alpha = H(A)$ (resp. $e = H(A)$) and now use it the same way we used α (resp. e) in DLSI-ZKS (resp. SRSA-ZKS). To answer a query x, open A as a, produce π_x and a MAC of Com under a. However note that both these variations (one-time signatures or MAC) cannot

[5] We can use the techniques from [13] to implement this step efficiently.

be used for strong independence as we need to sign several messages (C, x_i) with the same key.

It is possible to improve the efficiency of SRSA-ZKS by choosing N as product of two *safe* primes. In this case it is sufficient to use a collision-resistant hash function H that outputs small primes (smaller than $2^{k/2-1}$), making all the computations much faster.

It is also possible to obtain strong independence under the newly introduced Strong DDH assumption over Gap-DDH groups [3]. This approach uses the multi-trapdoor commitment from [13] based on this assumption, modified it to make it both mercurial and independent. Details will appear in the final version.

4 Independence versus Non-malleability for ZKS

In the previous section we showed that independence implies non-malleability for commitments. Does this implication extend to the case of ZKS protocols as well? The answer, surprisingly, is not that simple.

The first thing to clarify, of course, is a definition of non-malleability for ZKS protocols. Informally in the commitment case [11], a non-malleable commitment satisfies the following property. An adversary $\mathcal{A}$ is fed with a commitment to a message m, and she outputs another commitment to a message m'. If $\mathcal{A}$ manages to commit to a message m' related to m then there is another machine $\mathcal{A}'$ that outputs a commitment to m' without ever seeing a commitment to m. So in other words the commitment is not helping $\mathcal{A}$ in committing to related messages.

Our definition of non-malleability for ZKS follows the same paradigm. Except that, as in the case of independence, we have to deal with the fact that a ZKS commitment is a commitment to a large string and that the adversary may receive partial openings before creating her own commitment. For this reason we present three separate definitions, each stronger than the previous one and investigate their relationship with our notion of ZKS independence.

The discussion in this section is informal and intuitive. Formal definitions and proofs appear in Appendix J.

ZKS Weak Non-Malleability. A first attempt would be to consider ZKS protocols simply as commitments to large strings. In other words, as in the case of weak ZKS independence, the adversary commits *before* querying the honest provers. In this case the definition of ZKS non-malleability would be basically identical to non-malleability for commitment schemes.

Thus the proof of the following Corollary is basically identical to the proof of Theorem 1.

Corollary 1. *If a ZKS protocol is weakly ℓ-independent then it is weakly (ℓ, ϵ)-non-malleable with respect to opening.*

ZKS Non-Malleability. We can strengthen the above definition by allowing the adversary to query the committed databases before producing its own commitment, which must be different from the ones of the honest provers.

However now we are faced with a "selective decommitment problem" [12]. A ZKS commitment is a commitment to a large set of strings: by allowing the adversary to query some keys in the database we are basically allowing a selective decommitment of a subset of those strings (some points in the database).

Thus to obtain this form of ZKS non-malleability we need a commitment scheme which is secure against the selective decommitment problem. We do not know if independent or non-malleable commitments are secure in this sense. Universally composable (UC) commitments [4], on the other hand, are secure in the selective decommitment scenario.

However to obtain an efficient ZKS protocol, such UC commitments would have to be used inside the [19] construction, and thus would have to be mercurial as well. Unfortunately we do not know any commitment that is simultaneously mercurial and UC (not to mention also non-interactive).

Another approach is to restrict the distribution of the committed databases. Under this assumption we can prove that independence will suffice.

Let $\mathcal{IDB}$ be the family of distributions over databases where each distribution can be efficiently sampled conditioned on the value of some points in the database. In other words a distribution $\mathcal{DB} \in \mathcal{IDB}$ if after sampling $Db \in \mathcal{DB}$ and a set of points x_i it is possible to efficiently sample $Db' \in \mathcal{DB}$ such that Db, Db' agree on x_i. An example of such a class of distributions is the one in which the value of each element in the database is independent from the others.

Theorem 1 can easily be extended to the following.

Theorem 4. *If a ZKS protocol is ℓ-independent then it is (ℓ, ϵ)-non-malleable with respect to opening, with respect to the distribution class $\mathcal{IDB}$.*

ZKS Strong Non-Malleability. In this definition we allow the adversary to copy one of the commitments, of the honest provers. Now recall that when she is queried on her committed database, she can query the honest provers in the background on their databases. Since she copied the (say) i^{th} committed database, a distinguisher can always detect a correlation between the adversary's and P_i's answer to the same query x. But we require that this must be *all* that the distinguisher can see. In other words, the distinguisher cannot see any correlation between the answers of $\mathcal{A}$ and the answers of all the other P_j's; and cannot see any correlation between the answers of $\mathcal{A}$ and the answers of P_i unless it queries them on the same value.

Theorem 1 can also easily be extended to the following.

Theorem 5. *If a ZKS protocol is strongly ℓ-independent then it is strongly (ℓ, ϵ)-non-malleable with respect to opening, with respect to the distribution class $\mathcal{IDB}$.*

5 Open Problems

Can we build independent ZKS protocol based on general assumptions (the ones in this paper are based on either discrete log, Stong RSA or Strong DDH). A sufficient condition would be to build a commitment which is both independent and mercurial under general assumptions (the mercurial commitments from general assumptions in [16, 5] are not known to be independent).

Can we remove the restriction on the distribution of the input databases when we prove that independence implies non-malleability for ZKS? Here too a sufficient condition would be to build UC mercurial commitments.

In the case of commitments, is independence strictly stronger or equivalent to non-malleability?

References

1. J. Boyar, S.A. Kurtz and M.W. Krentel. *A Discrete Logarithm Implementation of Perfect Zero-Knowledge Blobs.* J. Cryptology 2(2): 63-76 (1990).
2. N. Barić, and B. Pfitzmann. *Collision-free accumulators and Fail-stop signature schemes without trees.* Proc. of EUROCRYPT '97, Springer LNCS 1233, pp.480-494.
3. D. Boneh and X. Boyen. *Short Signatures without Random Oracles.* Proc. of EUROCRYPT'04, Springer LNCS 3027, pp.382–400.
4. R. Canetti and M. Fischlin. *Universally Composable Commitments.* In Proc. of CRYPTO'01, Springer LNCS 2139, pp.19-40.
5. D. Catalano, Y. Dodis and I. Visconti. Mercurial Commitments: Minimal Assumptions and Efficient Constructions. To appear, TCC'06.
6. R. Cramer and I. Damgård. *New Generation of Secure and Practical RSA-based signatures.* Proc. of CRYPTO'96, Springer LNCS 1109, pp.173-185.
7. R. Cramer and V. Shoup. *Signature schemes based on the Strong RSA assumption.* ACM Transactions on Information and System Security (ACM TISSEC) 3(3):161-185, 2000.
8. I. Damgård, J. Groth. *Non-interactive and reusable non-malleable commitment schemes.* Proc. of 35^{th} ACM Symp. on Theory of Computing (STOC'03), pp.426-437, 2003.
9. G. Di Crescenzo, Y. Ishai, R. Ostrovsky. Non-Interactive and Non-Malleable Commitment. Proc. of 30^{th} ACM Symp. on Theory of Computing (STOC'98), pp.141–150, 1998.
10. G.Di Crescenzo, J. Katz, R. Ostrovsky, A. Smith. *Efficient and Non-interactive Non-malleable Commitment.* Proc. of EUROCRYPT 2001, Springer LNCS 2045, pp.40-59.
11. D. Dolev, C. Dwork and M. Naor. *Non-malleable Cryptography.* SIAM J. Comp. 30(2):391–437, 200.
12. C. Dwork, M. Naor, O. Reingold and L. Stockmeyer. *Magic Functions.* FOCS 1999.
13. R. Gennaro. *Multi-trapdoor Commitments and Their Applications to Proofs of Knowledge Secure Under Concurrent Man-in-the-Middle Attacks.* Proc. of CRYPTO'04, Springer LNCS 3152, pp.220–236.
14. S. Goldwasser, S. Micali and C. Rackoff. *The Knowledge Complexity of Interactive Proof Systems.* SIAM J. Comput. 18(1): 186-208 (1989)
15. S. Goldwasser, S. Micali, and R. Rivest. *A digital signature scheme secure against adaptive chosen-message attacks.* SIAM J. Computing, 17(2):281–308, April 1988.
16. M. Chase, A. Healy, A. Lysyanskaya, T. Malkin and L. Reyzin. *Mercurial Commitments and Zero-Knowledge Sets based on general assumptions.* Proc. of EUROCRYPT'05, Springer LNCS 3494, pp.422-439.

17. P. MacKenzie and K. Yang. *On Simulation-Sound Commitments.* Proc. of EUROCRYPT'04, Springer LNCS 3027, pp.382-400.
18. R.C. Merkle. *A Digital Signature Based on a Conventional Encryption Function.* Proc. of CRYPTO'87, Springer LNCS 293, pp.369–378.
19. S. Micali, M.O. Rabin and J. Kilian. *Zero-Knowledge Sets.* Proc. of FOCS'03, pp.80-91.
20. T. Pedersen. *Non-interactive and information-theoretic secure verifiable secret sharing.* Proc of CRYPTO'91, Springer LNCS 576, pp.129–140.
21. R. Ostrovsky, C. Rackoff and A. Smith. *Efficient Consistency Proofs for Generalized Queries on a Committed Database.* ICALP 2004.
22. R. Rivest, A. Shamir and L. Adelman. *A Method for Obtaining Digital Signature and Public Key Cryptosystems.* Comm. of ACM, 21 (1978), pp. 120–126

A Preliminaries

This section contains descriptions of all the preliminary tool we need for our results. It is quite long and can be skipped by readers familiar with the concepts described in here.

In the following we say that function $f(k)$ is negligible if for every polynomial $Q(\cdot)$ there exists an index k_Q such that for all $k > k_Q$, $f(k) \leq 1/Q(k)$.

In the course of the paper we will refer to probabilistic polynomial time Turing machines as *efficient* algorithms.

Also if $A(\cdot)$ is a randomized algorithm, with $a \leftarrow A(\cdot)$ we denote the event that A outputs the string a. With $Pr[A_1; \ldots ; A_k : B]$ we denote the probability of event B happening after $A_1, \ldots, A_k$.

A.1 Computational Assumptions

The Strong RSA Assumption. Let N be the product of two primes, $N = pq$. With $\phi(N)$ we denote the Euler function of N, i.e. $\phi(N) = (p-1)(q-1)$. With Z_N^* we denote the set of integers between 0 and $N-1$ and relatively prime to N.

Let e be an integer relatively prime to $\phi(N)$. The RSA Assumption [22] states that it infeasible to compute e-roots in Z_N^*. I.e. given a random element $s \in_R Z_N^*$ it is hard to find x such that $x^e = s \bmod N$.

The Strong RSA Assumption (introduced in [2]) states that given a random element s in Z_N^* it is hard to find $x, e \neq 1$ such that $x^e = s \bmod N$. The assumption differs from the traditional RSA assumption in that we allow the adversary to freely choose the exponent e for which she will be able to compute e-roots.

We now give formal definitions. Let $RSA(k)$ be the set of integers N, such that N is the product of two $k/2$-bit primes.

Assumption 1 *We say that the RSA Assumption holds, if for all probabilistic polynomial time adversaries $\mathcal{A}$ the following probability*

$$Pr[\ N \leftarrow RSA(k)\ ;\ e \leftarrow Z_{\phi(N)}^*\ ;\ s \leftarrow Z_N^*\ :\ \mathcal{A}(N,s,e) = x \ \ s.t.\ \ x^e = s \bmod N\ ]$$

is negligible in k. We say that the Strong RSA Assumption holds, if for all probabilistic polynomial time adversaries $\mathcal{A}$ the following probability

$$Pr[\ N \leftarrow RSA(k)\ ;\ s \leftarrow Z_N^*\ :\ \mathcal{A}(N,s) = (x,e)\ \ s.t.\ \ x^e = s \bmod N\ ]$$

is negligible in k.

The RSA Assumption can be strengthened to make it hold for a specific e, rather than a randomly chosen one.

A more efficient variant of our protocols requires that N is selected as the product of two *safe* primes, i.e. $N = pq$ where $p = 2p' + 1$, $q = 2q' + 1$ and both p', q' are primes. We denote with $SRSA(k)$ the set of integers N, such that N is the product of two $k/2$-bit safe primes. In this case the assumptions above must be restated replacing $RSA(k)$ with $SRSA(k)$.

Discrete Logarithm. Let G be a cyclic group of prime order q and let g be a generator of G. This means that for any element $y \in G$, there exists an unique $x \in Z_q$ such that $y = g^x$ in G. We say that the discrete logarithm is hard over G if it is infeasible to compute such x when given as input G, q, g, y.

More formally, we assume the existence of an efficient *istance generator* $\mathcal{I}$ that on input 1^k outputs (G, g, q) where G is a cyclic group generated by g and of prime order q where $|q| = k$.

Assumption 2 *We say that the discrete logarithm assumption holds over $\mathcal{I}$, if for all probabilistic polynomial time adversaries $\mathcal{A}$ the following probability*

$$Pr[\ (G,g,q) \leftarrow \mathcal{I}(1^k)\ ;\ x \leftarrow Z_q\ ;\ y \leftarrow g^x\ :\ \mathcal{A}(G,g,q,y) = x\ ]$$

is negligible in k.

A typical example of an istance generator is to select a random prime q of length k and then search for a prime p such that $q|(p-1)$. The cyclic group G is the subgroup of order q in Z_p^*.

A.2 Secure Signature Schemes

A secure signature scheme is a triple of algorithms (SG, Sig, Ver): SG is the key generation algorithm which on input a security parameter outputs (sk,vk) the signing/verification key pair; Sig is the (randomized) signature algorithm that on input a message and sk outputs a signature; Ver is the verification algorithm that on input a message/signature pair and vk accepts or rejects (always accepting pairs that were generated by Sig using sk).

We are going to use signature schemes which are secure against chosen message attack. Informally this means that the adversary is given the public key and the signature on messages of her choice (adaptively chosen). Then it is infeasible for the adversary to compute the signature of a different message. The following definition is from [15].

Definition 1. $(\mathsf{SG}, \mathsf{Sig}, \mathsf{Ver})$ *is a* secure signature *if for every probabilistic polynomial time forger $\mathcal{F}$, the following*

$$Pr\left[\begin{array}{c}(\mathsf{sk}, \mathsf{vk}) \leftarrow \mathsf{SG}(1^k)\ ;\ (M, \mathsf{sig}) \leftarrow \mathcal{F}^{\mathsf{Sig}(\mathsf{sk}, \cdot)}(\mathsf{vk})\ ; \\ \mathsf{Ver}(M, \mathsf{sig}, \mathsf{vk}) = 1 \ \ \textit{and } M \textit{ was not asked to the oracle}\end{array}\right]$$

is negligible in k.

A signature scheme is called *one-time* if the adversary $\mathcal{F}$ is limited to a single oracle query. One-time signatures can be constructed more efficiently than general signatures since they do not require public key operations.

A.3 Commitment Schemes

A commitment scheme is the equivalent of an opaque envelope. It is a two stage protocol: in the first part a sender commits to a message, and at the end of this stage, nobody else has any information about the message. The second stage is the opening stage, where the sender is bound to reveal the message he committed to: in other words either he does not answer, or he can only answer a single message, the one he committed to at the beginning.

More formally a (non-interactive) commitment scheme consists of the following algorithms:

- KG, key generation algorithm. On input the security parameter 1^k, it outputs pk, the public key.
- Com, the commitment algorithm. On input pk and a message M and R a random string in $\{0,1\}^k$, it computes $[C(M), D(M)] = \mathsf{Com}(\mathsf{pk}, M, R)$. $C(M)$ is the commitment string and is the output, while $D(M)$ is the opening of $C(M)$ and is kept secret.
- Ver, the verification algorithm. On input pk, a message M and two strings C, D it outputs 1 or 0.

We require the following properties. Assuming pk is chosen according to the distributions induced by KG:

Correctness For all messages M, if $[C(M), D(M)] = \mathsf{Com}(\mathsf{pk}, M, R)$, then $\mathsf{Ver}(\mathsf{pk}, M, C(M), D(M)) = 1$.

Secrecy Let $[C(M), D(M)] = \mathsf{Com}(\mathsf{pk}, M, R)$. We require that for every two messages M, M', the following distributions to be computationally indistinguishable: $C(M) \approx C(M')$.

Secure Binding Let $\mathcal{A}$ be an algorithm that on input pk wins if it outputs strings C, M, D, M', D' such that $M \neq M'$, and $\mathsf{Ver}(\mathsf{pk}, M, C, D) = \mathsf{Ver}(\mathsf{pk}, M', C, D') = 1$. Then for all efficient algorithms $\mathcal{A}$ the probability that $\mathcal{A}$ wins is negligible in the security parameter.

Equivocable Commitments Informally an equivocable commitment is one for which there exists a trapdoor that allows us to produce *fake* commitments with a distribution which is computationally indistinguishable from the one of honestly produced commitments. Knowledge of the trapdoor allows us to open fake commitments in any desired way.

Formally, a (non-interactive) equivocable commitment scheme is a commitment scheme as described above, except that

- The key generation algorithm KG, on input the security parameter 1^k, outputs a pair pk, tk where pk is the public key and tk is called the *trapdoor* (or secret key).
- There exists a fake commitment algorithm, $\mathsf{FakeCom}$, that on input pk,tk and R a random string in $\{0,1\}^k$, computes $C = \mathsf{FakeCom}(\mathsf{pk}, \mathsf{tk}, R)$.
- There exists an equivocation algorithm Equiv that opens a fake commitment in any possible way given the trapdoor information. It takes as input the key pair pk,tk, a fake commitment C, the random string R such that $C = \mathsf{FakeCom}(\mathsf{pk}, \mathsf{tk}, R)$, and an arbitrary message M. It outputs D such that $\mathsf{Ver}(\mathsf{pk}, M, C, D) = 1$.
- The **Secrecy** condition is modified as follows. Let $[C(M), D(M)] = \mathsf{Com}(\mathsf{pk}, M, R)$. We require that for every message M, the following distributions

  $$C(M) \approx \mathsf{FakeCom}(\mathsf{pk}, \mathsf{tk}, R) \quad \text{and} \quad D(M) \approx \mathsf{Equiv}(\mathsf{pk}, \mathsf{tk}, C, R, M)$$

  to be computationally indistinguishable[6] even when the distinguisher is given the trapdoor tk.

Trapdoor Commitments. *Trapdoor commitments* are a special type of equivocable commitments. In the case of trapdoor commitments, fake commitments are exactly the same as real ones (the two distributions are identical) and the secret information is not needed to create a fake commitment. A consequence of this fact, is that trapdoor commitments are information-theoretically secret (a property that may not hold for equivocable ones).

B Formal Definition of ZKS

A ZK Sets protocol is composed of the following algorithms:

- CRS, the common reference string generator algorithm, On input a security parameter 1^k it outputs $\sigma \in \{0,1\}^{k^c}$ for some constant c.
- $\mathsf{P1}$, the committing step of the Prover. On input the CRS σ and a database Db, $\mathsf{P1}$ outputs $[Com, Dec]$, the commitment/decommitment strings.
- $\mathsf{P2}$, the proving step of the Prover. On input σ, Db, Com, Dec and x, $\mathsf{P2}$ outputs π_x.

[6] Notice that the first condition implies that for any two messages M, M' the distributions $C(M)$ and $C(M')$ are computationally indistinguishable. The second condition states that a real opening of a real commitment is indistinguishable from an equivocation of a fake commitment

– V, the Verifier. On input σ, Com, x, π_x it outputs $y \in \{0,1\}^* \cup \{\perp, bad\}$.
– $\mathsf{Sim0}$,$\mathsf{Sim1}$, $\mathsf{Sim2}$, the simulators. $\mathsf{Sim0}$ runs on input the security parameter 1^k and outputs $\sigma' \in \{0,1\}^{k^c}$, and some internal state ω_0. $\mathsf{Sim1}$ runs on input σ', ω_0 and outputs Com' a commitment string and an update internal state ω. Finally $\mathsf{Sim2}$ runs on input σ', Com', ω and $x \in \{0,1\}^*$. It has access to an oracle $Db(\cdot)$ which returns the value $Db(x)$ for some database Db. $\mathsf{Sim2}$ returns π'_x.

The properties we require from these algorithms are the following:

Completeness For all databases Db and for all x

$$Pr\left[\begin{array}{c}\sigma \leftarrow \mathsf{CRS}(1^k)\ ;\ [Com, Dec] \leftarrow \mathsf{P1}(\sigma, Db);\\ \pi_x \leftarrow \mathsf{P2}(\sigma, Db, Com, Dec, x)\ ;\ \mathsf{V}(\sigma, Com, x, \pi_x) = Db(x)\end{array}\right] = 1$$

Soundness For all x and for all efficient algorithm $\mathsf{P'}$ we have that the following probability

$$Pr\left[\begin{array}{c}\sigma \leftarrow \mathsf{CRS}(1^k)\ ;\ [Com, \pi_x, \pi'_x] \leftarrow \mathsf{P}'(\sigma);\\ \mathsf{V}(\sigma, Com, x, \pi_x) = y\ ;\ \mathsf{V}(\sigma, Com, x, \pi'_x) = y';\\ y \neq y'\end{array}\right]$$

is negligible in k.

Zero-Knowledge For any efficient adversary $\mathcal{A}$, and for all efficiently computable[7] databases Db, the outputs of the following experiments are indistinguishable.

Real Experiment: $\sigma \leftarrow \mathsf{CRS}(1^k)$ and $[Com, Dec] \leftarrow \mathsf{P1}(\sigma, Db)$. For $i = 1$ to m: $\mathcal{A}(\sigma, Com, \pi_1, \ldots, \pi_{i-1})$ outputs x_i and gets $\pi_i = \mathsf{P2}(\sigma, Db, Com, Dec)$. The output is $[\sigma, x_1, \pi_1, \ldots, x_m, \pi_m]$.

Simulated Experiment: $(\sigma', \omega_0) \leftarrow \mathsf{Sim0}(1^k)$, $(Com', \omega) \leftarrow \mathsf{Sim1}(\omega_0)$. For $i = 1$ to m: $\mathcal{A}(\sigma', Com', \pi'_1, \ldots, \pi'_{i-1})$ outputs x_i and gets $\pi'_i = \mathsf{Sim2}^{Db}(\sigma', \omega, x_i)$. The output is $[\sigma', x_1, \pi'_1, \ldots, x_m, \pi'_m]$.

C Formal Definition of Mercurial Commitments

The following definition is due to Catalano *et al.* [5]:

– KG, the key generation algorithm. On input the security parameter 1^k, it outputs a key pair pk,tk, where pk is the public key which is put in the public record, while tk is kept secret.

[7] The original definition in [19] did not restrict Db to be efficiently computable, but in [16] it is pointed out that only statistical ZK can be achieved for those databases. We follow the definition in [16], but remind the reader that it is possible to prove this condition for all databases if one is proving statistical ZK. While the construction of ZK Sets from generic mercurial commitment achieves only computational ZK, the ones based on the specific number theoretic assumptions achieve statistical ZK, thus can be proven for any database.

– $\mathsf{HardCom}$, the hard commitment algorithm. On input pk and a message M it computes $[C(M), D(M)] = \mathsf{HardCom}(\mathsf{pk}, M, R)$ where R is a random string in $\{0,1\}^k$. As before $C(M)$ is the commitment string and $D(M)$ the opening.
– $\mathsf{SoftCom}$, the soft commitment algorithm. On input pk it computes $C = \mathsf{SoftCom}(\mathsf{pk}, R)$ where R is a random string in $\{0,1\}^k$.
– Tease, the (randomized) teasing algorithm. On input pk, M, C and R, where either $[C, D] = \mathsf{HardCom}(\mathsf{pk}, M, R)$, or $C = \mathsf{SoftCom}(\mathsf{pk}, R)$ it outputs a string τ.
– Ver, the hard verification algorithm. On input pk, a message M and two strings C, D it outputs 1 or 0.
– $\mathsf{VerTease}$, the tease verification algorithm. On input pk, a message M and two strings C, τ it outputs 1 or 0.

We require the following properties: the verification algorithm to be correct, that hard commitments preserve the secrecy of the message, that soft commitments be indistinguishable from hard ones, that hard commitments can be opened or teased in an unique way.

Below we describe more formally the above conditions. Assume pk is chosen according to the distributions induced by KG.

Correctness For all messages M, if $[C(M), D(M)] = \mathsf{HardCom}(\mathsf{pk}, M, R)$, then $\mathsf{Ver}(\mathsf{pk},M, C(M), D(M)) = 1$.
Also, for every message M, if $C = \mathsf{SoftCom}(\mathsf{pk}, R)$, and if $\tau = \mathsf{Tease}(\mathsf{pk},M, C, R)$ then $\mathsf{VerTease}(\mathsf{pk}, M, C, \tau) = 1$.

Secure Binding Let $\mathcal{A}$ be an algorithm that on input pk wins if it outputs strings C, M, D, M', D' such that $M \neq M'$, and $\mathsf{Ver}(\mathsf{pk},M, C, D) = \mathsf{Ver}(\mathsf{pk},M', C, D') = 1$ or $\mathsf{Ver}(\mathsf{pk},M, C, D) = \mathsf{VerTease}(\mathsf{pk},M', C, D') = 1$.
Then for all efficient algorithms $\mathcal{A}$ the probability that $\mathcal{A}$ wins is negligible in the security parameter.

Secrecy We define this condition differently that [16] but the definition is equivalent. There exist two algorithms $\mathsf{FakeCom}$ and Equiv, both taking as input the trapdoor tk: the first prepares "fake" commitments[8], while the second opens or teases fake commitments to any possible message.
  – $\mathsf{FakeCom}$ takes as input pk,tk and a random string R and outputs a fake commitment $C = \mathsf{FakeCom}(\mathsf{pk}, \mathsf{tk}, R)$;
  – Equiv takes as input pk,tk,C, R, such that $C = \mathsf{FakeCom}(\mathsf{pk}, \mathsf{tk}, R)$, a message M and either Open or Tease. It outputs a string D.

Let $[C(M), D(M)] = \mathsf{HardCom}(\mathsf{pk},M, R)$. We require the following distributions to be (computationally) indistinguishable even when the distinguisher is given the trapdoor tk:
  – for any message M, $C(M) \approx \mathsf{FakeCom}(\mathsf{pk}, \mathsf{tk}, R)$;
  – for any message M, $D(M) \approx \mathsf{Equiv}(\mathsf{pk}, \mathsf{tk}, C, R, M, \mathsf{Open})$;
  – $\mathsf{SoftCom}(\mathsf{pk},R) \approx \mathsf{FakeCom}(\mathsf{pk}, \mathsf{tk}, R)$;

[8] Note the important difference between fake and soft commitments: to prepare soft commitments one does not need the trapdoor tk.

– $\mathsf{Tease}(\mathsf{pk},M,C,R) \approx \mathsf{Equiv}(\mathsf{pk},\mathsf{tk},C,R,M,\mathsf{Tease})$;

The above basically says that hard and soft commitments are indistinguishable (since they are indistinguishable from fake commitments); for the same reason hard commitments preserve the secrecy of the message. Finally we are also requiring that the equivocations of fake commitments must be indistinguishable from either the opening or the teasing of hard/soft commitments (in particular this implies that the teasing of a hard commitment is indistinguishable from the teasing of a soft one).

D Detailed Description and Proof of MRK-ZKS

We refer to the protocol from [19] as MRK-ZKS = (MRK-CRS,MRK-P1,MRK-P2,MRK-V). A full detailed description follows.

For each node y in a binary tree denote with $left(y), right(y), sib(y)$ the left child, the right child and the sibling of y respectively.

MRK-ZKS = (MRK-CRS,MRK-P1,MRK-P2,MRK-V)

- MRK-CRS: the CRS is set to the public key pk for a trapdoor mercurial commitment scheme and H a collision resistant hash function.
- MRK-P1, the committing step of the prover works as follows. Build a tree of height l. Compute the subtree T as follows: let T' be the subtree composed by all the leaves x such that $x \in [Db]$, and the path connecting x to the root. Let $sib(T')$ be set of nodes y such that $sib(y) \in T'$. Then $T = T' \cup sib(T')$.
  For each leaf $x \in T$, if $x \in [Db]$ compute $[C_x, D_x] = \mathsf{HardCom}(\mathsf{pk},Db(x), r_x)$ for a random string r_x, and store C_x in leaf x. If $x \notin [Db]$ compute $C_x = \mathsf{SoftCom}(\mathsf{pk},r_x)$ for a random string r_x and store it in leaf x.
  Now for each level i in the T, starting from level $l-1$ (the parent of the leaves) up to level 1 (the root) do the following. For each node y at level i, if y has children then compute $[C_y, D_y] = \mathsf{HardCom}(\mathsf{pk},H(C_{left(y)}, C_{right(y)}), r_y)$ for a random r_y and store C_y in y. If y has no children in T then compute $C_y = \mathsf{SoftCom}(\mathsf{pk},r_y)$ for a random string r and store it in y.
  The commitment Com output by P1 is C_{root}. The decommitment information Dec is the collection of C_y, r_y used to compute the information at the internal nodes.
- MRK-P2, the proving step of the Prover, works as follows.
  On input $x \in [Db]$, the proof π_x consists of $Db(x)$ and the set of values $[C_y, D_y]$ for each node y in the path from the root to x (including x), and the values C_z for each node z such that $sib(z)$ is in the path.
  On input $x \notin [Db]$, let v be the last node on the path from the root to x that has a something stored in it. P2 augments the tree by computing for each node y such that y or $sib(y)$ is in the path from v to x (including x) the value $C_y = \mathsf{SoftCom}(\mathsf{pk},r_y)$ for a random string r_y and storing it in node y. Let $\tau_x = \mathsf{Tease}(\mathsf{pk},\perp, C_x, r_x)$. Also for each node y on the path from the root to x let $\tau_y = \mathsf{Tease}(\mathsf{pk},H(C_{left(y)}, C_{right(y)}), C_y, r_y)$.

Then the proof π_x consists of the set of values $[C_y, \tau_y]$ for each node y in the path from the root to x (including x), and the values C_z for each node z such that $sib(z)$ is in the path.
- MRK-V, the verifier works as follows. If C_{root} in π_x is different than C_{root} in Com it rejects immediately. Otherwise does the following.
On input x, π_x such that $x \in [Db]$, let $Db'(x)$ be the claimed value of $Db(x)$ contained in π_x. Then check that $\mathsf{Ver}(\mathsf{pk}, Db'(x), C_x, D_x) = 1$. Also check that for each node y in the path from the root to x (x not included) the following holds $\mathsf{Ver}(\mathsf{pk}, H(C_{left(y)}, C_{right(y)}), C_y, D_y) = 1$. If so output the value $Db(x) = Db'(x)$ otherwise output *bad*.
On input x, π_x such that $x \notin [Db]$. Check that $\mathsf{VerTease}(\mathsf{pk}, \perp, C_x, \tau_x) = 1$. Also check that for each node y in the path from the root to x (x not included) the following holds $\mathsf{VerTease}(\mathsf{pk}, H(C_{left(y)}, C_{right(y)}), C_y, \tau_y) = 1$. If so output $Db(x) = \perp$ otherwise output *bad*.

PROOF. **Completeness** should be obvious.

Soundness can be argued as follows. Assume there exists a Prover who is able to find x, π_x, π'_x such that the Verifier on input x, π_x outputs $Db(x) = y$ and on input x, π'_x outputs $Db(x) = y'$ with $y \neq y'$. If at least one of the y, y' is different from $\perp$ then this contradicts the *Secure Binding* condition of mercurial commitments. Indeed look at the nodes y on the path from the root to x and consider the highest node in which the two proofs agree on the value of C_y but its openings are different. Such a node must exist, because the openings of C_x in the two proofs are different, but the last commitment C_root must be the same in both proofs. Then the malicious prover has created a commitment C_y which can either be opened in two distinct ways or opened in one way and teased in another.

We now argue **Zero-Knowledge** by defining the simulators. MRK-Sim0 sets up the CRS so that it knows the matching tk for the public key pk. Then MRK-Sim1 using tk publishes a fake commitment $C_{root} = \mathsf{FakeCom}(\mathsf{pk}, \mathsf{tk}, r_{root})$.

For each query x, MRK-Sim2 obtains $Db(x)$ from his oracle and then simulates the Prover by just equivocating using tk. More specifically MRK-Sim2 performs the following steps:

- Let v be the highest node on the path from the root to x where there is something stored in the tree. If $v = x$ skip the next step.
- For each node y such that y or $sib(y)$ is in the path from x to v, if y is empty compute $C_y = \mathsf{FakeCom}(\mathsf{pk}, \mathsf{tk}, r_y)$ and store it in y.
- If $Db(x) = \perp$ then compute τ_y for each y in the path from the root to x as the real Prover using Equiv (in "Tease" mode).
- If $Db(x) \neq \perp$, use tk to run the algorithm Equiv (in "Open" mode) to compute the appropriate D_y for each node y in the path from the root to x. If a D_y has been computed before, use that.

Why is this view indistinguishable? Clearly the simulated CRS is equal to the real one. Also the commitment C_{root} is indistinguishable from the real CRS,C_{root}, by the *Secrecy* property of mercurial commitments.

The secrecy condition also guarantees that a simulated proof for a query x is indistinguishable from a real one, since simulated proofs are computed using the algorithm Equiv, whose output is indistinguishable from the real openings and teasings in real proofs.

E Independent Commitments

In this paper we focus on the notion of *independence with respect to opening* where it is required that the value opened by the adversary to be independent from the values opened by the honest senders. In the final paper we also present the notion of independence with respect to commitment (where independence is enforced at the time of adversary produces the commitment string).

For simplicity our definitions are presented in the common reference string model (in which a "public key" for the commitment scheme is generated in some way and put it in a public record to be used by all parties). Indeed all the known non-interactive non-malleable commitment schemes are in this model.

Our notion of independence can be defined only for *equivocable* commitment schemes, a concept that we have recalled in Appendix A and which includes the case of trapdoor commitments. Formally, we use the fact that our commitment schemes are equivocable, by feeding the adversary with fake commitments (which are indistinguishable from real ones) and then require that the adversary's behavior is the same no matter how we open them. A formal definition follows.

We say that an equivocable commitment scheme KG, Com, Ver, $\mathsf{FakeCom}$, Equiv is *ℓ-independent w.r.t. opening* if the following condition is satisfied.

ℓ-independence For any adversary $(\mathcal{A}_1, \mathcal{A}_2)$ and any pair of ℓ-tuple of messages $[M_1, \ldots, M_\ell]$, $[M'_1, \ldots, M'_\ell]$, the following probability is negligible in k:

$$Pr\left[\begin{array}{c} (\mathsf{pk},\mathsf{tk}) \leftarrow \mathsf{KG}(1^k)\ ;\ R_1,\ldots,R_\ell \leftarrow \{0,1\}^k\ ; \\ C_i \leftarrow \mathsf{FakeCom}(\mathsf{pk},\mathsf{tk},R_i)\ ; \\ (\omega, C) \leftarrow \mathcal{A}_1(\mathsf{pk}, C_1, \ldots, C_\ell)\ \text{ with }\ C \neq C_i\ \forall i\ ; \\ D_i \leftarrow \mathsf{Equiv}(\mathsf{pk},\mathsf{tk},C_i,R_i,M_i)\ ; \\ D'_i \leftarrow \mathsf{Equiv}(\mathsf{pk},\mathsf{tk},C_i,R_i,M'_i)\ ; \\ M, D \leftarrow \mathcal{A}_2(\mathsf{pk},\omega,M_1,D_1,\ldots,M_\ell,D_\ell)\ ; \\ M', D' \leftarrow \mathcal{A}_2(\mathsf{pk},\omega,M'_1,D'_1,\ldots,M'_\ell,D'_\ell)\ : \\ M \neq M' \text{ and } \mathsf{Ver}(\mathsf{pk},M,C,D) = \mathsf{Ver}(\mathsf{pk},M',C',D') = 1 \end{array}\right]$$

Let's first translate the meaning of the Definition in plain English. Consider the following process. Feed $\mathcal{A}_1$ with ℓ fake commitments, and wait for it to output a commitment of her own. At this point split the process in the two separate processes where we feed $\mathcal{A}_2$ with distinct openings of the ℓ commitments. Then the probability that $\mathcal{A}_2$ produces two *distinct* and *correct* openings is negligible. Thus if $\mathcal{A}_2$ decommits to a value, this value is unique.

We say that an equivocable commitment scheme is *independent* if it is ℓ-independent for any ℓ (polynomial in the security parameter).

F Proof of Theorem 1

For simplicity our definitions are presented in the common reference string model (in which a "public key" for the commitment scheme is generated in some way and put it in a public record to be used by all parties). Indeed all the known non-interactive non-malleable commitment schemes are in this model. First we recall the definition of non-malleability for commitments, and then we prove our main Theorem.

F.1 The definition of Non-Malleable Commitments

This section recalls the definition of non-malleability with respect to opening from [11, 9].

The informal, intuitive, security condition for non-malleable commitments, is that the adversary should not be able to commit to messages that are somewhat related to messages which are contained in commitments produced by honest parties. We are going to parameterize the definition by a value ℓ which is the number of commitments that the adversary sees from the honest parties[9].

Think of the following game. The adversary, after seeing a tuple of ℓ commitments produced by honest parties, outputs his own tuple of committed values. At this point the honest parties decommit their values and now the adversary tries to decommit his values in a way that his messages are related to the honest parties' ones. Intuitively, we say that a commitment scheme is ℓ-non-malleable (with respect to opening) if the adversary fails at this game.

However the adversary could succeed by pure chance, or because he has some a priori information on the distribution of the messages committed by the honest parties. So when we formally define non-malleability for commitments we need to focus on ruling out that the adversary receives any help from seeing the committed values. This can be achieved by comparing the behavior of the adversary in the above game, to the one of an adversary in a game in which the honest parties' messages are not committed to and the adversary must try to output related messages without any information about them.

Let's try to be more formal. We have a publicly known distribution $\mathcal{M}$ on the message space and a randomly chosen public key pk (chosen according to the distribution induced by KG).

Define Game 1 (the real game) as follows. We think of the adversary $\mathcal{A}$ as two separate efficient algorithms $\mathcal{A}_1, \mathcal{A}_2$. We choose ℓ messages according to this distribution, compute the corresponding commitments and feed them to the adversary $\mathcal{A}_1$. This adversary outputs a commitment, with the only restriction that he cannot copy any of the ℓ commitments presented to him. $\mathcal{A}_1$ also transfers some internal state ω to $\mathcal{A}_2$. We now open our commitments and run $\mathcal{A}_2$, who will the commitment prepared by $\mathcal{A}_1$ (if $\mathcal{A}_2$ refuses to open we replace the opening

[9] Surprisingly to prove that a commitment is 1-non-malleable (i.e. where the adversary sees only one honest commitment) is not sufficient to prove that it is ℓ-non-malleable (the intuition is that a standard hybrid argument would work).

with $\perp$). We then invoke a distinguisher $\mathcal{D}$ on the $\ell+1$ messages. $\mathcal{D}$ will decide if the two vectors are related or not (i.e. $\mathcal{D}$ outputs 1 if the messages are indeed related). We denote with $\mathsf{Succ1}_{\mathcal{D},\mathcal{A},\mathcal{M}}$ the probability that $\mathcal{D}$ outputs 1 in this game, i.e.

$$\mathsf{Succ1}_{\mathcal{D},\mathcal{A},\mathcal{M}}(k) = Pr\left[\begin{array}{c} \mathsf{pk} \leftarrow \mathsf{KG}(1^k)\ ;\ M_1,\ldots,M_\ell \leftarrow \mathcal{M}\ ; \\ R_1,\ldots,R_\ell \leftarrow \{0,1\}^k\ ;\ [C_i,D_i] \leftarrow \mathsf{Com}(\mathsf{pk},M_i,R_i)\ ; \\ (\omega,C) \leftarrow \mathcal{A}_1(\mathsf{pk},C_1,\ldots,C_\ell)\ \text{ with }\ C \neq C_i\ \forall i\ ; \\ M,D \leftarrow \mathcal{A}_2(\mathsf{pk},\omega,M_1,D_1,\ldots,M_\ell,D_\ell) \\ \text{s.t. } \mathsf{Ver}(\mathsf{pk},M,C,D)=1\ \text{ or }\ M=\perp\ ; \\ \mathcal{D}(M_1,\ldots,M_\ell,M)=1 \end{array}\right]$$

Define now Game 2 as follows. We still select ℓ messages according to $\mathcal{M}$ but this time feed nothing to the adversary $\mathcal{A}$. The adversary now has to come up with a message on its own. Again we feed the $\ell+1$ messages to $\mathcal{D}$ and look at the output. We denote with $\mathsf{Succ2}_{\mathcal{D},\mathcal{A},\mathcal{M}}$ the probability that $\mathcal{D}$ outputs 1 in this game, i.e.

$$\mathsf{Succ2}_{\mathcal{D},\mathcal{A},\mathcal{M}}(k) = Pr\left[\begin{array}{c} \mathsf{pk} \leftarrow \mathsf{KG}(1^k)\ ;\ M_1,\ldots,M_\ell \leftarrow \mathcal{M}\ ; \\ M \leftarrow \mathcal{A}(\mathsf{pk})\ \text{ s.t. }\ M \in \mathcal{M}\cup\{\perp\}\ ; \\ \mathcal{D}(M_1,\ldots,M_\ell,M)=1 \end{array}\right]$$

Finally we say that a distinguisher $\mathcal{D}$ is admissible, if for any input $(M_1,\ldots,M_\ell,M)$, its probability of outputting 1 does not increase if we change M into $\perp$. This prevents the adversary from artificially "winning" the game by refusing to open its commitments.

We say that the commitment scheme is (ℓ,ϵ)-*non-malleable* (with respect to opening) if for every message space distribution $\mathcal{M}$, every efficient admissible distinguisher $\mathcal{D}$, for every efficient adversary $\mathcal{A}$, and for every ϵ, there is an efficient adversary $\mathcal{A}'$ which runs in time polynomial in ϵ^{-1} such that the following difference

$$\mathsf{Succ1}_{\mathcal{D},\mathcal{A},\mathcal{M}}(k) - \mathsf{Succ2}_{\mathcal{D},\mathcal{A}',\mathcal{M}}(k) \leq \epsilon$$

In other words $\mathcal{A}'$ fares almost as well as $\mathcal{A}$ in outputting related messages.

We say that a commitment scheme is ϵ-*non-malleable* (with respect to opening) if it is (ℓ,ϵ)-non-malleable for any ℓ (polynomial in the security parameter).

F.2 The Proof of the Theorem

Before proving Theorem 1 let us stop for a second and compare the two definitions of independence and non-malleability. It should appear clear that the definition of independence is easier to work with and simpler. The implication from independence to non-malleability is intuitively immediate, yet requires a fair deal of technicalities. Virtually all the non-malleable commitments in the literature are independent (as we will point out later). Thus it is our hope that in the future, researchers can just focus on the easier to prove (and more immediately intuitive) goal of independence and then invoke Theorem 1 to claim non-malleability.

Proof. Let $\mathsf{KG}, \mathsf{Com}, \mathsf{Ver}, \mathsf{Equiv}$ be an ℓ-independent equivocable commitment scheme. Let $\mathcal{A} = [\mathcal{A}_1, \mathcal{A}_2]$ be an adversary for Game 1.

Consider the behavior of $(\mathcal{A}_1, \mathcal{A}_2)$ between Game 1 and an hybrid game which we call Game H. In Game H the ℓ commitments are prepared as fake commitments, and then opened to the messages M_i. Define the success probability of the adversary in Game H, the same way as in Game 1 and denote it with $\mathsf{SuccH}_{\mathcal{D},\mathcal{A},\mathcal{M}}(k)$. Since fake commitments are computationally indistinguishable from real ones, we have that the success probability of $(\mathcal{A}_1, \mathcal{A}_2)$ in Game H must be negligibly close to the success probability in Game 1 (i.e. $\mathsf{SuccH} \approx \mathsf{Succ1}$).

Now we want to build an adversary $\mathcal{A}'$ for Game 2 such that for every polynomial $P(\cdot)$ we have that

$$\mathsf{SuccH}_{\mathcal{D},\mathcal{A},\mathcal{M}}(k) - \mathsf{Succ2}_{\mathcal{D},\mathcal{A}',\mathcal{M}}(k) \leq \epsilon(k) = 1/P(k)$$

Let's fix such polynomial $P(\cdot)$ and consequently $\epsilon(k)$.

First of all $\mathcal{A}'$ runs $\mathsf{KG}(1^k)$ to obtain pk, tk. Then $\mathcal{A}'$ selects ℓ messages according to the distribution $\mathcal{M}$ and computes the fake commitments $C_1, \ldots, C_t$ accordingly. It then feeds the public key pk and the commitments to $\mathcal{A}_1$. Let ω, C be the output of $\mathcal{A}_1$.

Notice that up to this point, the view of $\mathcal{A}_1$ in this "simulated" game, is identical to the view that $\mathcal{A}_1$ would have had in Game H, no matter what the hidden messages $M_1, \ldots, M_\ell$ are. Thus the distribution of $\mathcal{A}_1$'s output will be the same in both games.

Now $\mathcal{A}'$ runs $k\epsilon^{-1}(k)$ copies of $\mathcal{A}_2$. On each copy, the internal state input will be set to the ω output by $\mathcal{A}_1$. However on each copy, $\mathcal{A}'$ will provide a different opening as follows. For each copy, $\mathcal{A}'$ samples ℓ messages from $\mathcal{M}$ and uses its knowledge of tk to open the commitments accordingly.

$\mathcal{A}'$ collects the openings of C that $\mathcal{A}_2$ provides in all the copies. $\mathcal{A}'$ sets M to the first such opening that is different from $\perp$. If in all the openings, $\mathcal{A}_2$ outputs $\perp$, then $\mathcal{A}'$ sets $M = \perp$.

Consider now the commitment C created by $\mathcal{A}_1$: we say that such a commitment is *bad*, if the probability that $\mathcal{A}_2$ opens it (i.e. opens to a non $\perp$ value) is less than $\epsilon(k)$; otherwise we say it is good. Since $\mathcal{A}'$ runs $k\epsilon^{-1}(k)$ copies of $\mathcal{A}_2$, we know that if C is good, then $\mathcal{A}'$ will get a correct opening with overwhelming probability. Notice that $\mathcal{A}'$ may also correctly open a bad commitment as well, but that does not matter as $\mathcal{D}$ is an admissible distinguisher.

Thus except than with probability negligibly close to $\epsilon(k)$, we have that (i) $\mathcal{A}_2$ does not open a bad commitment in the real life Game H; (ii) $\mathcal{A}'$ opens a good commitment in Game 2. So now all we have to prove is that $\mathcal{D}$ outputs 1 with almost the same probability on both outputs. But for $\mathcal{D}$ to output 1 with substantially more probability in Game H, rather than in Game 2, it means that for a good commitment, $\mathcal{A}_2$ must provide a different opening in real life Game H than in the simulation done during Game 2.

Said in other words, $\mathcal{A}_2$ opens C in two ways depending on what is the opening of the ℓ commitments presented to him. But by the property of independence this happens only with negligible probability.

G Weak ZKS Independence

In this section we define *weak ZKS independence* and present a generic ZKS protocol that is weakly independent.

Consider an adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ which tries to correlate its database to the one of a honest prover. $\mathcal{A}_1$ sees the commitments of the honest provers and outputs a commitment of its own. $\mathcal{A}_2$ is given concurrent access to the provers to whom he can ask several database queries while answering queries related to $\mathcal{A}_1$'s commitment. We would like these answers to be independent from the answers of the honest provers. Notice that $\mathcal{A}_1$ is not allowed to query the honest provers, thus its commitment is produced without any knowledge about the honest provers' databases.

In other words $\mathcal{A}_1$, after seeing the commitments $Com_1, \ldots, Com_\ell$ of ℓ honest provers, will output a commitment Com of its own. We now run two copies of $\mathcal{A}_2$, in the first we give him access to provers that "open" the Com_i according to some databases Db_i; in the second instead we use some different databases Db'_i. At the end $\mathcal{A}_2$ will outputs a value x and the corresponding proof π_x with respect to Com. We require that the database value associated to x in the two different copies of $\mathcal{A}_2$ must be the same, which implies that it is "fixed" at the end of the committing stage.

Of course we must rule out the adversary that simply copies the honest provers, as in that case achieving independence is not possible (or meaningful). Thus we require that $\mathcal{A}_1$ output a commitment Com different from the honest provers' Com_i.

A formal definition follows.

We say that a ZKS protocol is *ℓ-weakly-independent* if the following property holds:

ZKS ℓ-weak-independence For any adversary $(\mathcal{A}_1, \mathcal{A}_2)$ and for any pair of ℓ-tuple of databases $Db_1, \ldots, Db_\ell$ and $Db'_1, \ldots, Db'_\ell$ the following probability

$$Pr\left[\begin{array}{c} (\sigma, \omega_0) \leftarrow \mathsf{Sim0}(1^k)\ ;\ (Com_i, \omega_i) \leftarrow Sim1(\omega_0)\ \forall\ i = 1, \ldots, \ell\ ; \\ (Com, \omega) \leftarrow \mathcal{A}_1(\sigma, Com_1, \ldots, Com_\ell)\ \text{ with }\ Com \neq Com_i\ \forall i\ ; \\ (x, \pi_x) \leftarrow \mathcal{A}_2^{Sim2^{Db_i(\cdot)}(\omega_i, Com_i)}(\sigma, \omega)\ ; \\ (x, \pi'_x) \leftarrow \mathcal{A}_2^{Sim2^{Db'_i(\cdot)}(\omega_i, Com_i)}(\sigma, \omega)\ ; \\ bad \neq \mathsf{V}(\sigma, Com, x, \pi_x) \neq \mathsf{V}(\sigma, Com, x, \pi'_x) \neq bad \end{array}\right]$$

is negligible in k.

The above notion guarantees weak independence only if the adversary interacts with a bounded (ℓ) number of honest provers. We say that a ZKS protocol is *weakly independent* if it is ℓ-weakly-independent for any ℓ (polynomial in the security parameter). In this case weak independence is guaranteed in a fully concurrent scenario where the adversary can interact with as many honest parties as she wants.

THE GENERIC CONSTRUCTION. We now show a generic constructions for weakly independent ZKS. The construction use *any* ZKS protocol ZKS= (CRS, P1, P2, V) and *any* equivocable independent commitment Ind-Com = (KG, Com, Ver, FakeCom, Equiv). Notice that by the results in [9, 8, 17, 16] we obtain constructions of weak independent ZKS protocol based on general assumptions (collision-resistance hashing and trapdoor permutations). Here is the description of the first protocol:

WInd-ZKS:

- *CRS Generation.* The CRS is σ, pk where $\sigma \leftarrow \mathsf{CRS}(1^k)$ and $\mathsf{pk} \leftarrow \mathsf{KG}(1^k)$.
- *Prover's Committing Step.* On input Db: run P1 on σ, Db to obtain Com, Dec. Then choose $R \in_R \{0,1\}^k$ and compute $[C, D] = \mathsf{Com}(\mathsf{pk}, Com, R)$. The public commitment is C, the decommitting information is Com, Dec, D.
- *Prover's Proving Step.* On input x compute π_x with respect to Com using P2. Then output Com, D and π_x.
- *Verifier.* Check that Com, D is a correct decommitting of C under pk; if it is output $\mathsf{V}(\sigma, Com, x, \pi_x)$ otherwise output *bad*.

Theorem 6. *Assuming* Ind-Com *is an ℓ-independent commitment, and* ZKS *a zero-knowledge set protocol, then* WInd-ZKS *is an ℓ-weakly-independent zero-knowledge set protocol.*

An immediate consequence of the above is that if Ind-Com is independent, then WInd-ZKS is weakly independent.

Sketch of Proof: That WInd-ZKS is a ZKS protocol (if ZKS is) is obvious as we are running ZKS unchanged except for the extra commitment on top.

Let us elaborate on the simulator for WInd-ZKS as we are going to use it for the proof of independence. Let Sim0, Sim1, Sim2 be the simulators for ZKS. Here is the definition of our simulator for WInd-ZKS.

- Ind-Sim0. Run Sim0 to get σ, ω_0; run KG to get pk,tk. The values σ, pk are placed in the CRS while ω_0, tk is the internal state (trapdoor).
- Ind-Sim1. Run FakeCom (since it knows tk) to get a fake commitment C.
- Ind-Sim2. Run Sim1 (on input ω_0) to get Com, ω. Run Equiv to open C as Com, i.e. get D that represents a valid opening of C as Com. Keep Com fixed for all the queries x. On such a query run Sim2 on input Com, ω and oracle access Db to obtain π_x. Answer with Com, D, π_x.

This is clearly a valid ZK simulator, since we are basically running the simulator for ZKS, with the only difference that the commitment C is fake, and the opening D is an equivocation. But these are indistinguishable from real because Ind-Com is an equivocable commitment[10].

[10] Another simulator for WInd-ZKS would not use equivocation at all. It would run the simulators for ZKS, and compute the commitment and the opening using the real algorithms of Ind-Com (i.e. would not need to use the trapdoor tk.) However the simulator we chose allows us to postpone defining the value Com during the running of Sim2 which will be important for the proof of independence.

We need to prove ℓ-independence. Assume that there is an adversary $(\mathcal{A}_1, \mathcal{A}_2)$ that contradicts the ℓ-independence condition. That is we run $\mathsf{Ind\text{-}Sim0}$ to get σ, pk and ω_0, tk. Then we run ℓ copies of $\mathsf{Ind\text{-}Sim1}$ to get $C_1, \ldots, C_\ell$, i.e. ℓ fake commitments according to $\mathsf{Ind\text{-}Com}$. We feed these values to $\mathcal{A}_1$ which outputs $C \neq C_i$. Now we run two copies of $\mathcal{A}_2$ where on the first copy we answer its queries on C_i using $\mathsf{Ind\text{-}Sim2}$ with oracle access to Db_i, while in the second copy $\mathsf{Ind\text{-}Sim2}$ is given oracle access to Db'_i. Notice that in the two copies the opening of C_i will be different, say Com_i in the first, while Com'_i in the second (since $\mathsf{Ind\text{-}Sim2}$ runs independent copies of $\mathsf{Sim1}$ inside its execution).

At the end of this process $\mathcal{A}_2$ produces a value x and two "proofs" $(Com, D, \pi_x) \neq (Com', D', \pi'_x)$ in the two executions, such that the verifier will accept both and will output a different value for in the two executions. . Recall that Com, D is the opening of C according to $\mathsf{Ind\text{-}Com}$ with key pk. We have two cases according to the fact if $Com \neq Com'$ or $Com = Com'$.

Case 1. If $Com \neq Com'$, we break the independence of $\mathsf{Ind\text{-}Com}$. Indeed we have found two sequences of messages $Com_1, \ldots, Com_\ell$ and $Com'_1, \ldots, Com'_\ell$, such that if we equivocate C_i as Com_i or Com'_i the adversary opens its own C (which is different from all C_i's) in two different ways (all the extra things needed to run $\mathcal{A}_1, \mathcal{A}_2$ can be simulated).

Case 2. If $Com = Com'$ we contradicts the soundness of ZKS. The values D, D' are used by the verifier only to determine acceptance. Once the verifier accepts, its output is just a function of Com, π_x. Since the verifier accepts with two different outputs in the two executions, and $Com = Com'$ it must be that $\pi_x \neq \pi'_x$.

To break ZKS, we are given σ chosen according to CRS and we want to come up with Com, x, π_x, π'_x such that $bad \neq \mathsf{V}(\sigma, Com, x, \pi_x) \neq \mathsf{V}(\sigma, Com, x, \pi'_x) \neq bad$.

We change the above process in the following way: we use the σ that was given to us, rather than run the simulator. Clearly the view of $\mathcal{A}_1$ is indistinguishable from the one in the original process (simulated σ's are indistinguishable from real values). $\mathcal{A}_1$ will answer C.

We now create real ZKS commitments to the databases Db_i, Db'_i, call them Com_i, Com'_i. We run two copies of $\mathcal{A}_2$: in the first we equivocate C_i to Com_i and then answer queries according to the ZKS protocol. In the second we do the same except that we equivocate C_i to Com'_i. Again the joint view of $\mathcal{A}_2$ in the two executions is indistinguishable from the one in the above process.

$\mathcal{A}_2$ will come up with x and two different proofs $(Com, D, \pi_x) \neq (Com', D', \pi'_x)$. We are assuming $\pi_x \neq \pi'_x$. Also by hypothesis $bad \neq \mathsf{V}(\sigma, Com, x, \pi_x) \neq \mathsf{V}(\sigma, Com, x, \pi'_x) \neq bad$. So we can return Com, x, π_x, π'_x and break the soundness of ZKS. $\square$

Notice that the above protocol $\mathsf{WInd\text{-}ZKS}$ does not satisfy the stronger notion of independence, since if $\mathcal{A}_1$ is allowed to query the honest prover before committing, it will then see Com the ZKS commitment. Thus $\mathcal{A}_1$ may create Com' a commitment to a set related to the one contained in Com and proceed from there.

H Proof of DLSI-ZKS

Sketch of Proof: First of all, it is clear that DLSI-ZKS is a ZKS protocol: since we are just running MRK-ZKS with a specific mercurial commitment. Let us specify the behavior of the ZK simulator for DLSI-ZKS since it will be useful for the proof of independence.

- DLSI-Sim0. Compute $\sigma = (G, q, g_1, g_2, h, H)$, but with $g_2 = g_1^r$ and $h = g_1^s$. The value σ is placed in the CRS while r, s is the internal state (trapdoor).
- DLSI-Sim1. Choose sk,vk a key pair for a signature scheme. Compute $\alpha = H(\mathsf{vk})$ and $g_\alpha = g_1^\alpha g_2$. Run MRK-Sim1 using the mercurial commitment defined by $\sigma_\alpha = (G, q, g_\alpha, h)$. Notice that it can do that as it knows the "trapdoor" of such commitment (the discrete log $x_\alpha = (\alpha + r)s^{-1} \bmod q$ of h w.r.t. g_α).
  Recall that MRK-Sim1 produces a fake mercurial commitment Com. Output Com, vk.
- DLSI-Sim2. Run MRK-Sim2, using the mercurial commitment defined by $\sigma_\alpha = (G, q, g_\alpha, h)$. Also, on query x, append a signature on (Com, x) computed using sk.

This is basically the MRK-ZKS simulator, so the ZK property is guaranteed.

We need to prove strong 1-independence. Assume that there is an adversary $(\mathcal{A}_1, \mathcal{A}_2)$ and databases Db_1, Db_1' that contradict the strong 1-independence condition. That is we run DLSI-Sim0 to get σ and r, s. Then we run DSLI-Sim1 to get (Com_1, vk_1). Let $\alpha_1 = H(\mathsf{vk}_1)$ and $g_{\alpha_1} = g_1^{\alpha_1} g_2$. We feed (Com_1, vk_1), to $\mathcal{A}_1$ which queries some values on (Com_1, vk_1): we answer these by running DLSI-Sim2 with oracle access to Db_1. Let Q be the set of these queries. Then $\mathcal{A}_1$ outputs (Com, vk). Let $\alpha = H(\mathsf{vk})$ and $g_\alpha = g_1^\alpha g_2$.

Now we run two copies of $\mathcal{A}_2$ where on the first copy we answer its queries on (Com_1, vk_1) using DLSI-Sim2 with oracle access to Db_1, while in the second copy DLSI-Sim2 is given oracle access to $Db_1' \vdash_Q Db_1$ (this can be easily achieved by checking if the query is in Q or not and then using Db_1 or Db_1' accordingly).

At the end of this process the output of the first copy of $\mathcal{A}_2$ contains $(x, \pi_x, \mathsf{sig_x})$ while the second copy outputs $(x, \pi_x', \mathsf{sig}_x')$, such that the verifier will accept both strings and will output a different value for $Db(x)$ in the two executions.

Since the verifier accepts, sig_x and sig_x' must be valid signatures on Com, x under vk. We have several cases.

- *Case 1:* $\mathsf{vk} \neq \mathsf{vk}_1$. We show how to compute discrete log by breaking the soundness of MRK-ZKS. Assume we have a discrete log challenge G, q, g, y, i.e. we want to compute the discrete log of y w.r.t. g.
  We run the above process with the following modifications. DLSI-Sim0 chooses $\mathsf{sk}_1, \mathsf{vk}_1$ and sets $g_1 = g, g_2 = g^{-\alpha_1} y^r, h = y$ for $r \in_R Z_q$ and $\alpha_1 = H(\mathsf{vk}_1)$. With these settings DLSI-Sim1, DLSI-Sim2 know the discrete log of h w.r.t. g_{α_1} (the discrete log is $r^{-1} \bmod q$), so it can still run as before (using the $\mathsf{sk}_1, \mathsf{vk}_1$ selected by DLSI-Sim0).

$\mathcal{A}_2$ is outputting Com, x, π_x, π'_x which leads the verifier to accept and output 2 different database values for x: the soundness proof of MRK-ZKS shows how to use such proofs to compute the trapdoor of the mercurial commitment used in the protocol. In this case this means the discrete log z of h with respect to $g_\alpha = g_1^\alpha g_2$, where $\alpha = H(\mathsf{vk})$. By plugging in our settings for g_1, g_2, h we get the discrete log z of y with respect to $g^{\alpha-\alpha_1} y^r$. Thus

$$y = [g^{\alpha-\alpha_1} y^r]^z \implies y^{1-rz} = g^{\alpha-\alpha_1}$$

which means that the discrete log of y with respect to g is $(\alpha-\alpha_1)(1-rz)^{-1}$. Notice that by the collision-resistance of H, we have that $\alpha \neq \alpha_1$, thus $(1-rz) \neq 0$ and the inverse exists.

– *Case 2:* $\mathsf{vk} = \mathsf{vk}_1$. But in this case it must be that either $Com \neq Com_1$ or the value x was not asked by $\mathcal{A}_2$. Either way we break the signature scheme. We do the above simulation, but we set $\mathsf{vk}_1 = \mathsf{vk}$ where vk is the target key under which we want to forge. We have access to an oracle that signs messages for us, which we use to perform the simulation. On a query x_1 asked by $\mathcal{A}_2$ we need a signature on Com_1, x_1. We want to get a signature on a message that we did not ask to the oracle.
  By hypothesis $\mathcal{A}_2$ will use the same key vk, and will sign a message which is different from the ones we ask (since indeed either $Com \neq Com_1$ or the value x was not asked by $\mathcal{A}_2$).

□

I Proof of SRSA-ZKS

Sketch of Proof: The proof follows the same outline of the proof of Theorem 2. As before it is clear that SRSA-ZKS is a ZKS protocol: since we are just running MRK-ZKS with a specific mercurial commitment. Let us specify the behavior of the ZK simulator for SRSA-ZKS since it will be useful for the proof of independence.

– SRSA-Sim0. Compute $\sigma = (N, s, H)$, but store p, q such that $N = pq$. The value σ is placed in the CRS while p, q is the internal state (trapdoor).
– SRSA-Sim1. Choose sk,vk a key pair for a signature scheme. Compute $e = H(\mathsf{vk})$. Run MRK-Sim1 using the mercurial commitment defined by $\sigma_e = (N, s, e)$. Notice that it can do that as it knows the "trapdoor" of such commitment (the e-root of s, as it knows the factorization of N).
  Recall that MRK-Sim1 produces a fake mercurial commitment Com. Output Com, vk.
– SRSA-Sim2. Run MRK-Sim2, using the mercurial commitment defined by $\sigma_e = (N, s, e)$. Also, on query x, append a signature on (Com, x) computed using sk.

This is basically the MRK-ZKS simulator, so the ZK property is guaranteed.

We need to prove strong independence. Assume that there is an adversary $(\mathcal{A}_1, \mathcal{A}_2)$ and 2ℓ databases $Db_1, \ldots, Db_\ell$ and $Db'_1, \ldots, Db'_\ell$ that contradict the strong ℓ-independence condition for some ℓ. That is we run $\mathsf{SRSA\text{-}Sim0}$ to get σ and r, s. Then we run $\mathsf{SRSA\text{-}Sim1}$ to get $(Com_i, \mathsf{vk_i})$ for $i = 1, \ldots, \ell$. Let $e_i = H(\mathsf{vk_i})$. We feed $(Com_i, \mathsf{vk_i})$, to $\mathcal{A}_1$ which will first query some values on those commitments. We answer the queries to the i^{th} commitments by running $\mathsf{DLSI\text{-}Sim2}$ with oracle access to Db_i. Let Q_i be the set of these queries. Then $\mathcal{A}_1$ outputs (Com, vk). Let $e = H(\mathsf{vk})$.

Now we run two copies of $\mathcal{A}_2$ where on the first copy we answer its queries on $(Com_i, \mathsf{vk_i})$ using $\mathsf{SRSA\text{-}Sim2}$ with oracle access to Db_i, while in the second copy $\mathsf{SRSA\text{-}Sim2}$ is given oracle access to $Db'_i \vdash_{Q_i} Db_i$ (again this step can be performed by first checking if the query is in Q_i).

At the end of this process $\mathcal{A}_2$ produces a value x and two "proofs" (π_x, sig_x), $(\pi'_x, \mathsf{sig}'_x)$ in the two executions, such that the verifier will accept both and will output a different value in the two executions.

Since the verifier accepts, sig_x and sig'_x must be valid signatures on Com, x under vk. We have two cases.

- *Case 1:* $\mathsf{vk} \neq \mathsf{vk_i}$ for all i. We show how to break Strong RSA by breaking the soundness of $\mathsf{MRK\text{-}ZKS}$. Assume we have a strong RSA challenge N, δ and we want to compute $e \neq 1$ and $\delta^{1/e} \bmod N$.
  We run the above process with the following modifications. $\mathsf{SRSA\text{-}Sim0}$ chooses $\mathsf{sk_i}, \mathsf{vk_i}$. Let $e_i = H(\mathsf{vk_i})$ and set $s = \delta^{\prod_i e_i}$. We set the public key to N, s, H. With these settings $\mathsf{SRSA\text{-}Sim1}$, $\mathsf{SRSA\text{-}Sim2}$ know the e_i-roots of s so they can still run as before (using the $\mathsf{sk_i}, \mathsf{vk_i}$ selected by $\mathsf{SRSA\text{-}Sim0}$).
  $\mathcal{A}_2$ is outputting Com, x, π_x, π'_x which leads the verifier to accept and output 2 different database values for x: the soundness proof of $\mathsf{MRK\text{-}ZKS}$ shows how to use such proofs to compute the trapdoor of the mercurial commitment used in the protocol. In this case this means the e-root of s. Since $e \neq e_i$ for all i, we have that $GCD(e, \prod_i e_i) = 1$ and using standard techniques we get an e-root of δ.
- *Case 2:* $\mathsf{vk} = \mathsf{vk_1}$ for some i. But in this case it must be that either $Com \neq Com_i$ or the value x was not asked by $\mathcal{A}_2$ to the i^{th} prover. Either way we break the signature scheme.
  We do the above simulation, but we set $\mathsf{vk_i} = \mathsf{vk}$ where vk is the target key under which we want to forge for a randomly chosen i. We have access to an oracle that signs messages for us, which we use to perform the simulation. On a query x_i asked by $\mathcal{A}_2$ to the i^{th} prover we need a signature on Com_i, x_i. We want to get a signature on a message that we did not ask to the oracle. With probability $1/\ell$, $\mathcal{A}_2$ will use the same key vk, and will sign a message which is different from the ones we ask (since indeed either $Com \neq Com_i$ or the value x was not asked by $\mathcal{A}_2$ to the i^{th} prover).

$\square$

J Non-Malleability for ZKS Protocols

In this section we present definitions of non-malleability for ZKS protocols and then try to establish a relationship with our notion of independence.

Since we are going to talk about databases which are potentially large objects, in this section when we say that a distinguisher D is given a database Db as input we always mean that D is given oracle access to Db. Similarly with $\mathsf{V}^{\mathsf{P2}(Dec)}(\sigma, Com)$ we denote oracle access to the database contained in Com according to the verifier V: in other words on query x, the query is passed to $\mathsf{P2}$ who outputs π_x according to σ, Com, Dec following the ZKS protocol, and then V reaches the conclusion on the value $Db(x)$ by running on input σ, Com, x, π_x.

Weak Non-Malleability. A first attempt would be to consider ZKS protocols simply as commitments to large strings. In this case the definition would be basically identical to the case of commitment schemes. There is a known distribution $\mathcal{DB}$ from which databases are drawn.

In Game 1 the adversary $\mathcal{A}_1$ is given ℓ commitments to databases Db_i drawn from $\mathcal{DB}$. Then $\mathcal{A}_1$ produces a different commitment and some side information which is passed to $\mathcal{A}_2$. The distinguisher will query $\mathcal{A}_2$ on the commitment prepared by $\mathcal{A}_1$, while $\mathcal{A}_2$ is interacting with the honest provers. The distinguisher is also given as input the databases Db_i.

In Game 2 instead we feed $\mathcal{A}$ nothing (apart from the distribution $\mathcal{DB}$) and $\mathcal{A}$ will just output database queries for the distinguisher.

We require that the probability that the distinguisher outputs 1 in both games is roughly the same. More formally, let CRS, $\mathsf{P1}$, $\mathsf{P2}$, V be the ZKS Protocol. Denote with $\mathsf{Succ1w}_{\mathcal{D},\mathcal{A},\mathcal{DB}}$ the probability that $\mathcal{D}$ outputs 1 in Game 1, i.e.

$$\mathsf{ZKSucc1w}_{\mathcal{D},\mathcal{A},\mathcal{DB}}(k) = Pr\begin{bmatrix} \sigma \leftarrow \mathsf{CRS}(1^k) \ ; \ Db_1, \ldots, Db_\ell \leftarrow \mathcal{DB} \ ; \ (C_i, D_i) \leftarrow \mathsf{P1}(\sigma, Db_i) \ ; \\ (\omega, C) \leftarrow \mathcal{A}_1(\sigma, C_1, \ldots, C_\ell) \ \text{ with } \ C \neq C_i \ \forall i \ ; \\ \mathcal{D}(Db_1, \ldots, Db_\ell, \mathsf{V}^{\mathcal{A}_2(\omega)^{\mathsf{P2}(\sigma, C_i, D_i, \cdot)}}(\sigma, C)) = 1 \end{bmatrix}$$

We denote with $\mathsf{Succ2w}_{\mathcal{D},\mathcal{A},\mathcal{DB}}$ the probability that $\mathcal{D}$ outputs 1 in Game 2, i.e.

$$\mathsf{ZKSucc2w}_{\mathcal{D},\mathcal{A},\mathcal{DB}}(k) = Pr\begin{bmatrix} \sigma \leftarrow \mathsf{CRS}(1^k) \ ; \ Db_1, \ldots, Db_\ell \leftarrow \mathcal{DB} \ ; \ Db \leftarrow \mathcal{A} \ ; \\ \mathcal{D}(Db_1, \ldots, Db_\ell, Db) = 1 \end{bmatrix}$$

Finally we say that a distinguisher $\mathcal{D}$ is admissible, if for any $\ell + 1$ tuple of databases $Db_1, \ldots, Db_\ell, Db$, its probability of outputting 1 does not increase if the oracle queries to Db are answered by $\perp$. This prevents the adversary from artificially "winning" the game by refusing to open its commitments.

We say that the ZKS protocol is *weakly* (ℓ, ϵ)*-non-malleable* (with respect to opening) if for every database distribution $\mathcal{DB}$, every efficient admissible distinguisher $\mathcal{D}$, for every efficient adversary $\mathcal{A}$, and for every ϵ, there is an efficient adversary $\mathcal{A}'$ which runs in time polynomial in ϵ^{-1} such that the following difference

$$\mathsf{ZKSucc1w}_{\mathcal{D},\mathcal{A},\mathcal{DB}}(k) - \mathsf{ZKSucc2w}_{\mathcal{D},\mathcal{A}',\mathcal{DB}}(k) \leq \epsilon$$

In other words $\mathcal{A}'$ fares almost as well as $\mathcal{A}$ in outputting related databases.

A Stronger Definition. We can strengthen the above definition by allowing the adversary to query the committed databases before producing its own commitment, which must be different from the ones of the honest provers. The change is reflected in the definition of the probability $\mathsf{Succ1}$ where we need to give $\mathcal{A}_1$ access to the provers. Similarly definition of $\mathsf{Succ2}$ must reflect that the adversary is allowed to query some points on the databases.

$$\mathsf{ZKSucc1}_{\mathcal{D},\mathcal{A},\mathcal{DB}}(k) = Pr\left[\begin{array}{c}\sigma \leftarrow \mathsf{CRS}(1^k)\ ;\ Db_1,\ldots,Db_\ell \leftarrow \mathcal{DB}\ ;\ (C_i,D_i) \leftarrow \mathsf{P1}(\sigma,Db_i)\ ;\\ (\omega,C) \leftarrow \mathcal{A}_1^{\mathsf{P2}(\sigma,C_i,D_i,\cdot)}(\sigma,C_1,\ldots,C_\ell)\ \text{ with }\ C \neq C_i\ \forall i\ ;\\ \mathcal{D}(Db_1,\ldots,Db_\ell,\mathsf{V}^{\mathcal{A}_2(\omega)^{\mathsf{P2}(\sigma,C_i,D_i,\cdot)}}(\sigma,C)) = 1\end{array}\right]$$

$$\mathsf{ZKSucc2}_{\mathcal{D},\mathcal{A},\mathcal{DB}}(k) = Pr\left[\begin{array}{c}\sigma \leftarrow \mathsf{CRS}(1^k)\ ;\ Db_1,\ldots,Db_\ell \leftarrow \mathcal{DB}\ ;\ Db \leftarrow \mathcal{A}^{Db_i}\ ;\\ \mathcal{D}(Db_1,\ldots,Db_\ell,Db) = 1\end{array}\right]$$

We say that the ZKS protocol is (ℓ,ϵ)-*non-malleable* (with respect to opening) with respect to a class of database distributions $\{\mathcal{DB}\}$, if for every $\mathcal{DB}$ in the class, every efficient admissible distinguisher $\mathcal{D}$, for every efficient adversary $\mathcal{A}$, and for every ϵ, there is an efficient adversary $\mathcal{A}'$ which runs in time polynomial in ϵ^{-1} such that the following difference

$$\mathsf{ZKSucc1}_{\mathcal{D},\mathcal{A},\mathcal{DB}}(k) - \mathsf{ZKSucc2}_{\mathcal{D},\mathcal{A}',\mathcal{DB}}(k) \leq \epsilon$$

In other words $\mathcal{A}'$ fares almost as well as $\mathcal{A}$ in outputting related databases.

Notice that we are basically handling the problem of non-malleability with selective decommitment described in the previous Section. A ZKS commitment is a commitment to a large set of strings, and the adversary is allowed a selective decommitment of a subset of those strings (some points in the database). As for the case of single messages, we do not know how to obtain non-malleability in this case with respect to *any* distribution over the databases.

Let $\mathcal{IDB}$ be the family of distributions over databases where each distribution can be efficiently sampled conditioned on the value of some points in the database. In other words a distribution $\mathcal{DB} \in \mathcal{IDB}$ if after selecting $Db \in \mathcal{DB}$ and a set of points x_i it is possible to efficiently sample $Db' \in \mathcal{DB}$ such that Db, Db' agree on x_i. An example of such a class of distributions is the one in which the value of each element in the database is independent from the others.

The strongest definition. In this definition we allow the adversary to copy one of the commitments. But then we restrict the distinguisher. We say that a distinguisher is i-admissible if the intersection of the queries to the i^{th} and $(\ell+1)^{st}$ database is empty. We define

$$\mathsf{ZKSucc1s}_{\mathcal{D},\mathcal{A},\mathcal{DB}}(k) = Pr\left[\begin{array}{c}\sigma \leftarrow \mathsf{CRS}(1^k)\ ;\ Db_1,\ldots,Db_\ell \leftarrow \mathcal{DB}\ ;\ (C_i,D_i) \leftarrow \mathsf{P1}(\sigma,Db_i)\ ;\\ (\omega,C) \leftarrow \mathcal{A}_1^{\mathsf{P2}(\sigma,C_i,D_i,\cdot)}(\sigma,C_1,\ldots,C_\ell);\\ \mathcal{D}(Db_1,\ldots,Db_\ell,\mathsf{V}^{\mathcal{A}_2(\omega)^{\mathsf{P2}(\sigma,C_i,D_i,\cdot)}}(\sigma,C)) = 1\end{array}\right]$$

where if $C = C_i$ for some i then $\mathcal{D}$ must be i-admissible. We also define $\mathsf{ZKSucc2s}_{\mathcal{D},\mathcal{A},\mathcal{DB}}(k)$ identically as $\mathsf{ZKSucc2}$ above.

We say that the ZKS protocol is *strongly* (ℓ,ϵ)-*non-malleable* (with respect to opening) with respect to a class of database distributions $\{\mathcal{DB}\}$, if for every

$\mathcal{DB}$ in the class, every efficient admissible distinguisher $\mathcal{D}$, for every efficient adversary $\mathcal{A}$, and for every ϵ, there is an efficient adversary $\mathcal{A}'$ which runs in time polynomial in ϵ^{-1} such that the following difference

$$\mathsf{ZKSucc1s}_{\mathcal{D},\mathcal{A},\mathcal{DB}}(k) - \mathsf{ZKSucc2s}_{\mathcal{D},\mathcal{A}',\mathcal{DB}}(k) \leq \epsilon$$

In other words $\mathcal{A}'$ fares almost as well as $\mathcal{A}$ in outputting related databases.