Mercurial Commitments with Applications to Zero-Knowledge Sets*

Melissa Chase† Alexander Healy‡ Anna Lysyanskaya§ Tal Malkin¶ Leonid Reyzin‖

January 6, 2009

Abstract

We introduce a new flavor of commitment schemes, which we call *mercurial commitments*. Informally, mercurial commitments are standard commitments that have been extended to allow for *soft* decommitment. Soft decommitments, on the one hand, are not binding but, on the other hand, cannot be in conflict with true decommitments.

We then demonstrate that a particular instantiation of mercurial commitments has been implicitly used by Micali, Rabin and Kilian to construct *zero-knowledge sets*. (A *zero-knowledge set* scheme allows a Prover to (1) commit to a set S in a way that reveals nothing about S and (2) prove to a Verifier, in zero-knowledge, statements of the form $x \in S$ and $x \notin S$.) The rather complicated construction of Micali et al. becomes easy to understand when viewed as a more general construction with mercurial commitments as an underlying building block.

By providing mercurial commitments based on various assumptions, we obtain several different new zero-knowledge set constructions.

1 Introduction

1.1 Mercurial Commitments

A traditional cryptographic commitment is often compared to a safe. The sender places a particular value in the safe, locks it and gives it to the recipient. The recipient cannot see the value, but is assured that it will not change while inside the safe. Then, whenever the sender chooses to, he can reveal the secret code needed to open the safe, enabling the recipient to retrieve the hidden value. Therefore, the two usual requirements of commitment are that it be *binding* and *hiding*: the sender is bound to the message, but the message is hidden from the recipient.

We propose a variant of traditional commitments, where the opening protocol is two-tiered. Partial opening, which we call "teasing", is not truly binding: it is possible for the sender to come up with a commitment that can be teased to any value of the sender's choice. True opening, on the other hand, is binding in the traditional (computational) sense: it is infeasible for the sender to come up with a commitment that he can open to two different values.

Despite the fact that a commitment can potentially be teased to any value, a tease is not merely a meaningless assertion. A tease of a commitment to a value m is a guarantee that the commitment

*A preliminary version of this work appeared in Eurocrypt 2005 [CHL+05].

†Microsoft Research, Redmond, WA 98052. email: melissac@microsoft.com. Much of this work done while at Brown University.

‡Division of Engineering and Applied Sciences, Harvard University, Cambridge, MA 02138. email: ahealy@post.harvard.edu.

§Department of Computer Science, Brown University, Providence, RI 02912. email: anna@cs.brown.edu.

¶Department of Computer Science, Columbia University, New York, NY 10027. email: tal@cs.columbia.edu.

‖Department of Computer Science, Boston University, Boston, MA 02215. email: reyzin@cs.bu.edu.

cannot be opened to any value other than m. In other words, the recipient of a tease knows that if the commitment can be opened at all, then it will be to the same value. It is infeasible for the sender to come up with a commitment that can be teased to m_1 and opened to $m_2 \neq m_1$.

This immediately implies, of course, that if the sender can open a commitment at all, then it can be teased to only one value. Thus, the sender must choose, at the time of commitment, whether to "soft-commit," so as to be able to tease to multiple values but not open at all, or to "hard-commit," so as to be able to tease and to open to only one particular value. The recipient, however, cannot tell which of the two options the sender has chosen (this is ensured by the hiding property).

We call this new primitive *mercurial commitment*.

Mercurial commitments are different from trapdoor or chameleon commitments of [BCC88]. All chameleon commitments are equivocal whenever the sender knows a trapdoor for the commitment scheme. In mercurial commitments, on the other hand, the sender is given the choice, at the time of commitment, whether to make the commitment equivocal or binding. Furthermore, in chameleon commitments, equivocated and regular decommitments look the same to the recipient; whereas in mercurial commitments, the recipient may be content with the decommitment that may have been equivocated (tease), or may require the stronger full decommitment (open).

Note that mercurial commitments directly imply conventional commitments as a special case, when only hard-commit and open are used (and the soft-commit and tease functionalities are ignored).

We have not yet addressed the hiding property of mercurial commitments. For our application, we need a very strong hiding property, namely, simulatability (which we can provide in the shared random string or trusted parameters model[1], or else interactively). However, such strong hiding does not seem to be an essential property of mercurial commitments, and it is conceivable that, if mercurial commitments find other applications, weaker hiding properties will suffice. Indeed, subsequent to the publication of [CHL$^+$05], mercurial commitments with weaker hiding (based on indistinguishability rather than simulatability) were considered by Dodis, Catalano, and Visconti [CDV06]. To distinguish our strong hiding property, subsequent work has used the adjective "trapdoor" for mercurial commitments that possess it (while we find this terminology helpful, we do not use it here because trapdoor mercurial commitments are the only ones we define).

We briefly describe our notation in Section 2.1, and then formally define mercurial commitments in Section 2.2. We provide four constructions in Section 2.3: based on general (possibly noninteractive) zero-knowledge, claw-free permutations, discrete logarithms, and factoring respectively. The last two constructions are efficient enough to be useable in practice. Finally, in Section 3, we describe the application to zero-knowledge sets.

Subsequent to our work, Catalano, Dodis, and Visconti showed how to construct mercurial commitments from one-way functions in the shared random string model [CDV06, Section 4]. They have also provided a generalization (in the trusted parameters model) of our second construction [CDV06, Section 3] as well as some new constructions and a framework to unify them with our third and fourth constructions [CDV06, Section 5]. A way to modify mercurial commitments to allow for updates was proposed by Liskov [Lis05]. Mercurial commitments that possess independence (a property implying nonmalleability) were constructed in [GM06]. A generalization of mercurial commitments allowing for committing to an ordered sequence of messages was proposed by Catalano, Fiore, and Messina [CFM08].

We distilled the notion of mercurial commitments out of the zero-knowledge set construction of [MRK03], where a particular construction (namely, the one based on discrete logarithms) of mercurial commitments is implicitly used. We believe that abstracting this notion and separating its construction from

[1]The shared random string model assumes that a uniform random string is available for all parties to use. The trusted parameters model assumes that a public string from some (possibly complex) distribution has been produced and is available for all parties to use; furthermore the (uniform) coins used to produce that string are unknown to the parties (for instance, such a string could be a product n of two large primes p and q, where the primes themselves are unknown to the parties).

the construction of zero-knowledge sets themselves is beneficial. First, as we demonstrate in Section 3.2, the [MRK03] construction of zero-knowledge sets becomes conceptually simpler when mercurial commitments are used as a building block. Second, when mercurial commitments can be studied in isolation, it is much easier to come up with novel constructions for them, and therefore also for zero-knowledge sets. Finally, mercurial commitments may be interesting in their own right.

1.2 Zero-Knowledge Sets

Zero-knowledge sets (ZKS) were recently introduced by Micali, Rabin, and Kilian [MRK03]. ZKS allow a prover to commit to an arbitrary finite set S in such a way that for any string x he can provide an efficient sound proof of whether $x \in S$ or $x \notin S$, without revealing any knowledge beyond this membership assertion. That is, the recipient (verifier) of the proof learns nothing else about the set S, not even the size of S. We elaborate on the formal definition of ZKS in Section 3.1.

As pointed out by [MRK03], the notion of zero-knowledge sets can be extended to zero-knowledge elementary databases, where each element $x \in S$ has a value $v(x)$ associated with it. After committing to S, the prover can provide an efficient proof for each x of either "$x \in S$ and $v(x) = v$", or "$x \notin S$", without revealing any further information. Sets, of course, are a special case of this, where the value associated with each $x \in S$ is 1. Throughout this paper, we use ZKS to refer also to the more general zero-knowledge elementary databases.

Micali, Rabin, and Kilian give a construction of zero-knowledge sets under the discrete logarithm assumption in the shared random string model. This construction is noninteractive (i.e., both the initial commitment and query answers require a single message from the prover to the verifier) with $O(k^2)$-bit proofs for security parameter k. They do not show how to remove the number-theoretic details of their construction, and leave open whether constructions not based on the discrete logarithm assumption are possible at all.

It is an interesting problem to consider what alternative constructions are possible, and under what assumptions these constructions can be realized.

Ostrovsky, Rackoff and Smith [ORS04] provide constructions for consistent database queries, which allow the prover to commit to a database, and then provide query answers that are provably consistent with the commitment. They also consider the problem of adding privacy to such protocols. Their constructions can handle queries much more general than just membership queries; they yield two constructions of ZKS as special cases. The first construction is a feasibility result, showing that interactive ZKS can be built out of (public) collision-resistant hash functions (CRHF) and zero-knowledge proofs of NP statements (which require only one-way functions, which are implied by CRHF); noninteractive ZKS can be built in the shared random string model out of CRHF and noninteractive zero-knowledge. The second construction is more efficient, based on the assumptions of CRHF and homomorphic commitments. Unfortunately, it requires interaction (which can be removed in the random oracle model) and requires the prover to keep a counter t of the number of queries asked so far. (For security parameter k, the proofs are of size $O(tk^4)$ and, in particular, grow with t.[2])

We provide an alternative proof of the same feasibility result, as well as more efficient constructions based on different assumptions, as detailed next.

1.3 Zero-Knowledge Sets from Mercurial Commitments

We describe the work of [MRK03] in light of our new primitive, thus showing how to construct zero-knowledge sets based on mercurial commitments and collision-resistant hash functions. Different in-

[2]The proof size given in [ORS04] is $O(tdsk^2)$, where s is a bound on the length of each key x, and d is a bound on logarithm of the set size. However, in order to hide the set size, we must first hash each key to a k-bit value, and set $d = k$.

stantiations of mercurial commitments will result in different ZKS constructions with different security assumptions and efficiency.

Instantiating our ZKS construction with mercurial commitments based on general zero-knowledge gives an alternative proof of the feasibility of ZKS from general assumptions (as mentioned above, another such proof was given independently by [ORS04]). It shows that (noninteractive) ZKS can be constructed in the shared random string model by using as building blocks noninteractive zero-knowledge (NIZK) proofs [BDMP91, FLS99], (conventional) commitment schemes (which are already implied by NIZK), and CRHF. If one is willing to add interaction to the revealing (membership proof) phase of ZKS, our construction shows that CRHF and interactive ZKS are equivalent (because NIZK can be replaced with regular zero-knowledge proofs, which can be based on one-way functions, which are implied by CRHF; on the other hand, it is quite clear that CRHF is necessary for ZKS, because the initial commitment to the set must be collision-resistant). Theorem 4 of [CDV06] subsequently showed that interaction is not needed, and ZKS in the shared random string model is equivalent to CRHF. Unfortunately, the above discussion applies merely to feasibility results; none of these constructions is practical.

Instantiating our ZKS construction with mercurial commitments based on claw-free permutations gives ZKS in the trusted parameters model with proof length $O(k^3)$. The construction based on factoring further improves the efficiency, giving ZKS with proof length $O(k^2)$ and verification time $O(k^4)$, suitable for practical implementation in the trusted parameters model.

For the case of ZKS from discrete-logarithm-based mercurial commitments (which are the ones implicitly used in [MRK03]), we provide a constant-factor improvement over the [MRK03] construction by utilizing a hash function better suited for such commitments. The resulting construction is within the realm of practical implementation in the shared random string model, requiring proofs of length $O(k^2)$ and verification time $O(k^4)$ (constants hidden by O here are fairly small and are further analyzed in Section 3.3.2).

Subsequent work on mercurial commitments provided additional constructions of ZKS based on different assumptions and with different features, performance profiles, and security properties [Lis05, CDV06, GM06, CDV06]. Prabhakaran and Xue [PX07] proposed a variant of ZKS with a relaxed security definition and showed a construction that is not based on mercurial commitments.

2 The New Primitive: Mercurial Commitments

2.1 GMR Notation

Following Micali, Rabin, and Kilian [MRK03], we use the GMR notation [GMR88, Mic]. This section is almost verbatim from [Lys02a].

Let A be an algorithm. By $A(\cdot)$ we denote that A has one input (resp., by $A(\cdot, \ldots, \cdot)$ we denote that A has several inputs). By $y \leftarrow A(x)$, we denote that y was obtained by running A on input x. If A is deterministic, then this y is unique; if A is probabilistic, then y is a random variable. If S is a finite set, then $y \leftarrow S$ denotes that y was chosen from S uniformly at random. By $y \in A(x)$ we mean that the probability that y is output by $A(x)$ is positive.

By $A^O(\cdot)$, we denote a Turing machine that makes queries to an oracle O. I.e., this machine will have an additional (read/write-once) query tape, on which it will write its queries in binary; once it is done writing a query, it inserts a special symbol "#". By external means, once the symbol "#" appears on the query tape, oracle O is invoked and its answer appears on the query tape adjacent to the "#" symbol. By $Q = Q(A^O(x)) \leftarrow A^O(x)$ we denote the contents of the query tape once A terminates, with oracle O and input x. By $(q, a) \in Q$ we denote the event that q was a query issued by A, and a was the answer received from oracle O.

By $(\mathtt{state}, y) \leftarrow A(\mathtt{state}, x)$ we denote that A is a *stateful* algorithm, and the variable $\mathtt{state}$ denotes its state information. We will write $(\mathtt{state}_A, y) \leftarrow B^{A(\mathtt{state}_A, \cdot)(x)}$ to denote that B has oracle access to a stateful algorithm A.

Let b be a boolean function. By $(y \leftarrow A(x) : b(y))$, we denote the event that $b(y)$ is true after y was generated by running A on input x. The statement $\Pr[\{x_i \leftarrow A_i(y_i)\}_{1 \leq i \leq n} \ : \ b(x_n)] = \alpha$ means that the probability that $b(x_n)$ is TRUE after the value x_n was obtained by running algorithms $A_1, \ldots, A_n$ on inputs $y_1, \ldots, y_n$, is α, where the probability is over the random choices of the probabilistic algorithms involved.

A function $\nu(k)$ is *negligible* if for every positive polynomial $p(\cdot)$ and for sufficiently large k, $\nu(k) < \frac{1}{p(k)}$.

2.2 Definition

As we describe in the introduction, a mercurial commitment is a commitment scheme with additional features. The first feature is that, in addition to the usual algorithm for opening a commitment, there is also an algorithm to partially open, or *tease*. The partial decommitment of a commitment C to a value x means, in essence, that if C can be opened at all, then it can be opened only to x. The second feature of a mercurial commitment scheme is that a commitment C can be formed in two ways: it may be a *hard* commitment, that is, a commitment that can be opened (and teased) in only one way; or a *soft* commitment that cannot be opened at all, but can be teased to any value. Let us now describe this more formally.

A mercurial commitment scheme in the shared random string or trusted parameters model consists of the following algorithms:

Setup This is a randomized algorithm run by a trusted third party that sets up the public key for the commitment. We write $PK \leftarrow \textsc{Setup}(1^k)$. The chosen public key PK defines the (efficiently samplable) domain of possible committed values. Let us denote this domain D_{PK}. If this algorithm merely outputs its random coins, then the mercurial commitment scheme is in the *shared random string* model. Else it is in the stronger *trusted parameters* model.

Hard-Comm This is the *deterministic* algorithm used to commit to a value. It takes as input the public key PK, a value $x \in D_{PK}$, and a random string r, and outputs the commitment C. We write $C = \textsc{Hard-Comm}(PK, x, r)$.

Soft-Comm This is the *deterministic* algorithm used to soft-commit. That is to say, a value produced by this algorithm is not really a commitment because it can never be opened. But it can be partially opened (teased) to any value of the committer's choice. This algorithm takes as input the public key PK and the random string r, and outputs a value C. We write $C = \textsc{Soft-Comm}(PK, r)$.

Tease This is the randomized algorithm for partially opening (teasing) a hard or soft commitment. On input (PK, x, r, C), where C is either a hard commitment to x with string r, or a soft commitment with string r, Tease outputs the partial decommitment τ for teaser value x. We write $\tau \leftarrow \textsc{Tease}(PK, x, r, C)$.

Ver-Tease This is the algorithm that either accepts or rejects the partial decommitment τ to teaser value x. It takes as input the public key PK, the commitment C, and the values x and τ.

Open This algorithm opens the commitment C. If $C = \textsc{Hard-Comm}(PK, x, r)$, then on input (PK, x, r, C), Open will output the decommitment π for the committed value x. We write $\pi \leftarrow \textsc{Open}(PK, x, r, C)$.

Ver-Open This is the algorithm that either accepts or rejects the decommitment π to the value x. It takes as input the public key PK, the commitment C, and the values x and π.

As usual for commitment schemes, we require three properties: (1) correctness: Ver-Tease will always accept the correctly formed partial decommitment τ of C for the correct teaser value x, and Ver-Open will always accept the correctly formed decommitment π of C for the correct x; (2) binding: no adversary can create C such that it can be opened to two different values, and no adversary can create C such that it can be opened to one value but partially decommitted (teased) to another value; (3) hiding: no adversary can learn whether C is a soft commitment or hard commitment, and in case it is a hard commitment, no adversary can learn the committed value x; moreover, we require that there be a *simulator* that will be able to form C in such a way that it can later not only partially decommit (tease) it to any teaser value, but also open it to any value, so that the view of the receiver will be the same whether it is talking to the committer or to the simulator.

More precisely:

Definition 2.1. A set of algorithms Setup, Hard-Comm, Soft-Comm, Tease, Open, Ver-Tease and Ver-Open satisfies the *correctness* property of mercurial commitments if for all $PK \in \text{Setup}(1^k)$

- Correctness for Hard-Comm: For all $x \in D_{PK}$, for all strings r, if $C = \text{Hard-Comm}(PK, x, r)$, then
  - for all $\tau \in \text{Tease}(PK, x, r, C)$, $\text{Ver-Tease}(PK, C, x, \tau) = ACCEPT$;
  - for all $\pi \in \text{Open}(PK, x, r)$, $\text{Ver-Open}(PK, C, x, \pi) = ACCEPT$;
- Correctness for Soft-Comm: For all r, if $C = \text{Soft-Comm}(PK, r)$, then for all $x \in D_{PK}$, for all $\tau \in \text{Tease}(PK, x, r, C)$, $\text{Ver-Tease}(PK, C, x, \tau) = ACCEPT$.

Definition 2.2. A set of algorithms Setup, Ver-Tease and Ver-Open satisfies the *binding* property of mercurial commitments if for all probabilistic polynomial-time nonuniform adversaries $\{A_k\}$ there exists a negligible function ν such that

$$\begin{aligned}
\Pr[PK \leftarrow \text{Setup}(1^k); (C, x, x', \pi, \tau) \leftarrow A_k(PK) \ :& \\
\text{Ver-Open}(PK, C, x, \pi) = ACCEPT \wedge& \\
(\text{Ver-Open}(PK, C, x', \tau) = ACCEPT \vee \text{Ver-Tease}(PK, C, x', \tau) = ACCEPT) \wedge& \\
x \neq x' \in D_{PK}] \quad = \quad \nu(k)&
\end{aligned}$$

Definition 2.3. A set of algorithms Setup, Hard-Comm, Soft-Comm, Tease, Open satisfies the *hiding (simulatability)* property of mercurial commitment if

1. There exists a set of algorithms Sim-Setup, Sim-Commit, Sim-Tease, Sim-Open with the following specifications:

   Sim-Setup This is a randomized algorithm that, in addition to creating the commitment public key PK, also outputs a trapdoor key TK that allows the simulator some extra power that the legitimate committer does not have. We write $(PK, TK) \leftarrow \text{Sim-Setup}(1^k)$.

   Sim-Commit This is the deterministic algorithm that the simulator uses to compute a commitment. Besides (PK, TK), it takes a random string r as input. We write $C = \text{Sim-Commit}(PK, TK, r)$.

   Sim-Tease This is the algorithm that the simulator uses to compute a partial decommitment for any value $x \in D_{PK}$. On input (PK, TK, r, x), it gives the partial decommitment τ for the commitment $C = \text{Sim-Commit}(PK, TK, r)$. We write $\tau \leftarrow \text{Sim-Tease}(PK, TK, r, x)$.

   Sim-Open This is the algorithm that the simulator uses to compute a decommitment for any value $x \in D_{PK}$. On input (PK, TK, r, x), it outputs the decommitment π for the commitment $C = \text{Sim-Commit}(PK, TK, r)$. We write $\pi \leftarrow \text{Sim-Open}(PK, TK, r, x)$.

2. Define the following algorithms:

Committer$_{\boldsymbol{PK}}$ The committer algorithm C_{PK} is a stateful algorithm that responds to requests to hard- and soft-commit to specific values by running Hard-Comm and Soft-Comm, and then, on request runs the Tease and Open algorithms on the corresponding commitments. It also maintains a list L of commitments issued so far. Initially, list L is empty. Here is how C_{PK} responds to various inputs:

- On input $(\textsc{Hard-Comm}, x)$, choose a random string r. Compute $C = \textsc{Hard-Comm}(PK, x, r)$. Store $(\textsc{Hard-Comm}, C, x, r)$ on the list L. Output C.
- On input $(\textsc{Soft-Comm})$, choose a random string r. Compute $C = \textsc{Soft-Comm}(PK, r)$. Store $(\textsc{Soft-Comm}, C, r)$ on the list L. Output C.
- On input $(\textsc{Tease}, C, x')$:
  - Check if and entry containing commitment C is stored on the list L. If it is not, output "fail." Else, retrieve the record corresponding to C.
  - If C's entry on the list is of the form $(\textsc{Hard-Comm}, C, x, r)$: if $x \neq x'$, output "fail." Otherwise, output $\tau = \textsc{Tease}(PK, x, r, C)$.
  - Else if C's entry on the list is of the form $(\textsc{Soft-Comm}, C, r)$: output $\tau = \textsc{Tease}(PK, x', r, C)$.
- On input $(\textsc{Open}, C, x)$, check if for some r, $(\textsc{Hard-Comm}, C, x, r)$ is on the list. If it is not, output "fail." Else, output $\textsc{Open}(PK, x, r)$.

Simulator$_{(\boldsymbol{PK},\boldsymbol{TK})}$ The simulator $S_{(PK,TK)}$ answers queries of the same types as the Committer C_{PK}, but by running different algorithms. It also maintains the list L *just as C_{PK} does.* The list L is initialized to empty.

- On input $(\textsc{Hard-Comm}, x)$, choose a random string r. Compute $C = \textsc{Sim-Commit}(PK, TK, r)$. Store $(\textsc{Hard-Comm}, C, x, r)$ on the list L. Output C.
- On input $(\textsc{Soft-Comm})$, choose a random string r. Compute $C = \textsc{Sim-Commit}(PK, TK, r)$. Store $(\textsc{Soft-Comm}, C, r)$ on the list L. Output C.
- On input $(\textsc{Tease}, C, x')$:
  - Check if any entry containing the commitment C is stored on the list L. If it is not, output "fail." Else, retrieve the record corresponding to C.
  - If C's entry on the list is of the form $(\textsc{Hard-Comm}, C, x, r)$: if $x \neq x'$, output "fail." Otherwise, output $\tau \leftarrow \textsc{Sim-Tease}(PK, TK, x, r, C)$.
  - Else if C's entry on the list is of the form $(\textsc{Soft-Comm}, C, r)$: output $\tau \leftarrow \textsc{Sim-Tease}(PK, TK, x', r, C)$.
- On input $(\textsc{Open}, C, x)$, check if for some r, $(\textsc{Hard-Comm}, C, x, r)$ is on the list. If it is not, output "fail." Otherwise, output $\textsc{Sim-Open}(PK, TK, x, r)$.

Then no polynomial-time distinguisher can tell whether he is talking to a Committer or to a Simulator. Namely, for all probabilistic polynomial-time families of oracle Turing machines $\{D_k^?\}$, there exists a negligible function $\nu(k)$ such that

$$\begin{aligned} \Pr[PK_0 \leftarrow \textsc{Setup}(1^k); (PK_1, TK) \leftarrow \textsc{Sim-Setup}(1^k);& \\ O_0 = C_{PK_0}; O_1 = S_{(PK_1,TK)};& \\ b \leftarrow \{0,1\}; b' \leftarrow D_k^{O_b}(pk_b) \ : \ b = b'] &\ = \ 1/2 + \nu(k) \end{aligned}$$

(In this definition, we create two oracles: O_0 is a Committer, and O_1 is a Simulator. Then the distinguisher interacts with a randomly chosen oracle, and has to guess which oracle it is talking to.)

Note that, while this definition does not explicitly state that hard commitments are indistinguishable from soft commitments, it is directly implied: The simulator uses SIM-COMMIT(PK, TK, r) both when answering HARD-COMM queries and when answering SOFT-COMM queries, so the resulting distributions will be identical. An adversary who could distinguish hard commitments generated by HARD-COMM(PK, x, r) from soft commitments generated by SOFT-COMM(PK, x, r) would thus be able to distinguish interaction with the Committer from interaction with the Simulator.

Remarks. Note that the notion of simulatability can be defined in three flavors: perfect, statistical, and computational, corresponding to the strength of the distinguisher D. Above, we gave the definition for the computational flavor since it is the least restrictive. Similarly, the binding property can be strengthened to statistical or even perfect to give security against computationally unbounded adversaries. Also note that the definition above is noninteractive. The definition can be extended to an interactive setting; while an interactive commitment phase seems of little applicability to zero-knowledge sets, constructions with interactive decommitments (opening or teasing) may be used in ZKS-like schemes. Throughout the paper, in order to keep the presentation clean, we continue by default to consider noninteractive mercurial commitments (and noninteractive ZKS), and mention the interactive case only in side remarks.

Definition 2.4 (mercurial commitment scheme)**.** Algorithms SETUP, HARD-COMM, SOFT-COMM, TEASE, OPEN, VER-TEASE and VER-OPEN constitute a mercurial commitment scheme if they satisfy the correctness, binding, and hiding (simulatability) properties.

Subsequent to our work, alternative definitions were proposed by Catalano, Dodis, and Visconti [CDV06, Section 2.3]. One alternative considers indistinguishability-based, rather than our simulatability-based, hiding (and is called *plain* mercurial commitment as opposed to *trapdoor* mercurial commitment). Another alternative, which we will call the CDV definition (see paragraph entitled "(Trapdoor) Mercurial Commitments" in [CDV06, Section 2.3]) is very similar to ours, but, by slightly strengthening the adversary, can be formulated more simply and thus allows for simpler proofs. The strengthening is the following: the SETUP and SIM-SETUP algorithms are required to be the same, and, in particular, there is always a trapdoor for the public key. The simulated commitments/openings are required to be indistinguishable from real ones even given the trapdoor.

Our constructions from Sections 2.3.2, 2.3.3, and 2.3.4 satisfy the CDV definition, while the construction in Section 2.3.1 does not, but can be modified to do so depending on the underlying NIZK system. The most notable example of a trapdoor mercurial commitment scheme construction not satisfying the CDV definition is the construction from [CDV06, Section 4], which shows that one-way functions imply mercurial commitments in the shared random string model. However, the CDV definition is satisfied by the construction from [CDV06, Section 3], which gives the same result in the trusted parameters model.

2.3 Constructions

2.3.1 From General Assumptions

We construct mercurial commitments based on a many-theorem noninteractive zero-knowledge proof system [BDMP91, FLS99]. Such a proof system can be constructed from any trapdoor permutation (TDP) [BDMP91, BY96], or from a verifiable random function (VRF) [GO92, MRV99]. Existence of TDPs and existence of VRFs are, as far as we know, incomparable assumptions, since VRFs can be constructed based on gap-Diffie-Hellman groups [Lys02b], while no trapdoor permutation is known based on such groups. This construction is in the shared random string model (not in the trusted parameters model).

Suppose that we are given a many-theorem noninteractive zero-knowledge (NIZK) proof system for an NP-complete language L. This proof system operates in the shared random string model, and consists of polynomial-time algorithms Prove and Verify. Further suppose that we are given a conventional noninteractive unconditionally binding commitment scheme, consisting of algorithms (Comm-Setup, Commit). Note that such a commitment scheme is already implied by the existence of NIZK, because NIZK implies one-way functions, and in the shared random string model, one-way functions imply setup-free unconditionally binding bit commitment [Nao91, HILL99]. More detailed definitions of these standard building blocks, NIZK and commitment schemes, are given in Appendix A.

Below, we describe a (noninteractive) mercurial commitment scheme based on a NIZK proof system and any noninteractive commitment scheme. The idea of this construction is simple: a mercurial commitment will consist of two conventional commitments. The first one determines whether it is a hard-commit or soft-commit. The second one determines the value itself in case of hard-commit. To tease, simply prove (using NIZK) that "either this is a soft-commit, or the committed value is x." To open, prove (using NIZK) that "this is a hard-commit to x." Correctness will follow from the correctness properties of the NIZK and of the commitment scheme. The binding property will follow from the fact that the commitment scheme is unconditionally binding, and from the soundness of the NIZK. Simulatability will follow from the security of the commitment scheme and from the zero-knowledge property (i.e., existence of the zero-knowledge simulator) of the NIZK.

More formally, consider the following construction of a mercurial commitment scheme:

Building blocks NIZK proof system (Prove, Verify) for all of NP, unconditionally binding commitment scheme (Comm-Setup, Commit).

Setup On input 1^k:

- Generate a random string σ for the NIZK proof system with security parameter k. Let us say that the NIZK proof system requires an $\ell(k)$-bit random string. Then $\sigma \leftarrow \{0,1\}^{\ell(k)}$.
- Run the setup algorithm for the commitment scheme to obtain the public parameters required: $p \leftarrow \text{Comm-Setup}(1^k)$.
- Output $PK = (\sigma, p)$. (Let the domain D_{PK} be the same as for the underlying commitment scheme. Without loss of generality, assume that $0, 1 \in D_{PK}$.)

Hard-Comm On input (PK, x, r):

- Parse $PK = (\sigma, p)$ and r as two random strings, r_1 and r_2.
- Compute $C_1 = \text{Commit}(p, 0, r_1)$, $C_2 = \text{Commit}(p, x, r_2)$; output commitment $C = (C_1, C_2)$.

Soft-Comm On input (PK, r):

- Parse $PK = (\sigma, p)$ and r as two random strings, r_1 and r_2.
- Compute $C_1 = \text{Commit}(p, 1, r_1)$, $C_2 = \text{Commit}(p, 1, r_2)$; output commitment $C = (C_1, C_2)$.

Tease On input (PK, x, r, C):

- Parse $PK = (\sigma, p)$, $C = (C_1, C_2)$, and $r = r_1 \circ r_2$.
- Let the NP-language L_t be defined as follows:

$$\begin{aligned} L_t &= \{X : X = (p, x, C_1, C_2) \text{ such that } \exists r_1, r_2 \\ &\qquad \text{such that } C_1 = \text{Commit}(p, 1, r_1) \vee C_2 = \text{Commit}(p, x, r_2)\} \end{aligned}$$

Let $X = (p, x, C_1, C_2)$. Let the witness $W = (r_1, r_2)$. Compute and output $\tau = \text{Prove}(\sigma, X, W)$.

VER-TEASE On input (PK, C, x, τ):

- Parse $PK = (\sigma, p)$ and $C = (C_1, C_2)$; let $X = (p, x, C_1, C_2)$
- Output the decision of $\textsc{Verify}(\sigma, X, \tau)$ for language L_t defined above.

OPEN On input (PK, x, r, C):

- Parse $PK = (\sigma, p)$, $C = (C_1, C_2)$, and $r = r_1 \circ r_2$.
- Let the NP-language L_o be defined as follows:

$$\begin{aligned} L_o = \{X : X = (p, x, C_1, C_2) \text{ such that } \exists r_1, r_2 \\ \text{such that } C_1 = \textsc{Commit}(p, 0, r_1) \wedge C_2 = \textsc{Commit}(p, x, r_2)\} \end{aligned}$$

Let $X = (p, x, C_1, C_2)$. Let the witness $W = (r_1, r_2)$. Compute and output $\pi = \textsc{Prove}(\sigma, X, W)$ for language L_o. [3]

VER-OPEN On input (PK, C, x, π):

- Parse $PK = (\sigma, p)$ and $C = (C_1, C_2)$; let $X = (p, x, C_1, C_2)$
- Output the decision of $\textsc{Verify}(\sigma, X, \pi)$ for language L_o defined above.

Theorem 2.1. *The construction above is a mercurial commitment scheme, assuming the underlying primitives satisfy the definitions of NIZK proofs for NP and unconditionally binding commitment schemes.*

Proof. The correctness property is clear. Let us prove the binding and the simulatability properties.

The binding property The binding property follows by a standard argument from the soundness property of the noninteractive zero-knowledge proof system. We give this argument here. Suppose that we have an adversary that, on input $PK = (\sigma, p)$ can, with nonnegligible probability $\epsilon(k)$, produce values C, x, x' and π and π' such that $x \neq x'$, $\textsc{Ver-Open}(PK, C, x, \pi) = ACCEPT$ and $\textsc{Ver-Open}(PK, C, x', \pi') = ACCEPT$. By construction this means that $C = (C_1, C_2)$ are such that π and π' are, respectively, NIZK proofs that (1) there exist r_1 and r_2 such that $C_1 = \textsc{Commit}(p, 0, r_1)$ and $C_2 = \textsc{Commit}(p, x, r_2)$; and (2) there exist r'_1 and r'_2 such that $C_1 = \textsc{Commit}(p, 0, r'_1)$ and $C_2 = \textsc{Commit}(p, x', r'_2)$. With high probability over the choice of p, no such r_2 and r'_2 exist, and so one of the statements (1) or (2) is false. Therefore, if the NIZK verifier accepts the proofs π and π', then the proof system is not sound. The case when the adversary produces the values C, x, x' and π and τ are such that $x \neq x'$, $\textsc{Ver-Open}(PK, C, x, \pi) = ACCEPT$ and $\textsc{Ver-Tease}(PK, C, x', \tau) = ACCEPT$, is analogous.

The simulatability property Let us give the descriptions of the algorithms that comprise the simulator, and then prove that the resulting simulation works.

Building blocks The simulator algorithms for the NIZK proof system (ZKSIM-SETUP, ZKSIM-PROVE). The algorithm ZKSIM-SETUP outputs a shared string σ (indistinguishable from random) as well as some secret value s required for simulating a noninteractive zero-knowledge proof. ZKSIM-PROVE takes an input the string σ that was generated using ZKSIM-SETUP, the secret s and some true statement $x \in L$ where L is a language in NP, and outputs a proof π that is distributed in such a way that it is indistinguishable from a proof that would be produced by the real prover $\textsc{Prove}(\sigma, x, w)$ where w is any witness for $x \in L$.

[3] Note that since both L_o and L_t are in NP, we can prove statements in both of these languages using the same random string σ, as long as we have multi-theorem proof system designed for some NP-complete language. (E.g. the proof systems in [FLS99], [BDMP91], etc.)

SIM-SETUP On input 1^k:

- Compute $(\sigma, s) \leftarrow$ ZKSIM-SETUP(1^k).
- Run the setup algorithm for the commitment scheme and obtain the public parameters required: $p \leftarrow$ COMM-SETUP(1^k).
- Output $PK = (\sigma, p)$, $TK = s$.

SIM-COMMIT On input (PK, TK) and some randomness (r_1, r_2), parse $PK = (\sigma, p)$, compute $C_1 =$ COMMIT$(p, 0, r_1)$, $C_2 =$ COMMIT$(p, 0, r_2)$, and output $C = (C_1, C_2)$.

SIM-TEASE On input $(PK, TK, (r_1, r_2), x, (C_1, C_2))$, parse $PK = (\sigma, p)$, $TK = s$, and for the language L_t defined above, output $\tau \leftarrow$ ZKSIM-PROVE$(\sigma, s, (p, x, C_1, C_2))$.

SIM-OPEN On input $(PK, TK, (r_1, r_2), x, (C_1, C_2))$, parse $PK = (\sigma, p)$, $TK = s$ and for the language L_o defined above, output $\pi \leftarrow$ ZKSIM-PROVE$(\sigma, s, (p, x, C_1, C_2))$.

What remains to be shown is that no distinguisher will be able to tell whether he is talking to a committer or to a simulator. This follows by a standard hybrid argument. Let us construct "hybrid" oracles whose output is distributed "in between" the outputs of the Committer and the Simulator.

Hybrid H_{C0} This oracle first sets up the public parameters according to the correct SETUP algorithm. It then behaves in exactly the same way as the Committer oracle.

Hybrid H_{C1} This oracle sets up the public parameters according to the SIM-SETUP algorithm. However, from that point on it behaves the same way as the Committer oracle.

Hybrid H_0 This oracle sets up the public parameters according to the SIM-SETUP algorithm. It executes all the HARD-COMM and SOFT-COMM queries correctly. However, when responding to an OPEN or TEASE query, it uses ZKSIM-PROVE instead of PROVE for computing a noninteractive zero-knowledge proof. Note that this is the same as running SIM-TEASE and SIM-OPEN instead of TEASE and OPEN.

Hybrid H_i This oracle sets up the public parameters according to the SIM-SETUP algorithm. For the first i commit queries, it runs SIM-COMMIT instead of HARD-COMM and SOFT-COMM. Beginning from HARD-COMM or SOFT-COMM query number $i+1$, this oracle starts producing the correct output. This oracle always uses SIM-TEASE and SIM-OPEN in place of TEASE and OPEN.

Let $T(D)$ be a polynomial bound on the number of queries issued by D. It is clear that Hybrid $H_{T(D)}$ acts the same way that the Simulator does. Therefore, it is sufficient to show that (1) H_{C0} is indistinguishable from H_{C1}; (2) H_{C1} is indistinguishable from H_0; and (3) for all polynomials poly(k), for all $0 < i \leq \text{poly}(k)$, Hybrid H_i behaves in a way that is indistinguishable from Hybrid H_{i-1}.

Statement (1) follows from the fact that the output σ of ZKSIM-SETUP for NIZK is indistinguishable from a random σ. Similarly, statement (2) follows because we are using a multi-theorem noninteractive zero-knowledge proof system, and so ZKSIM-PROVE's output is distributed the same as that of PROVE.

Finally, for (3) we need a (standard) hybrid argument. Namely, suppose that for some i, H_{i-1} can be distinguished from H_i. Then we break the security of the commitment scheme COMMIT. We attack COMMIT as follows: suppose that we are given public parameters p, and two commitments C_1 and C_2. The task is to tell whether C_1, C_2 are commitments to $(0, 0)$ or not. Further, suppose that we have a distinguisher D distinguishing between H_{i-1} and H_i. Then we run him as follows: first, we generate the public parameters: output $PK = (\sigma, p)$, where σ is the output of ZKSIM-SETUP. Note that this is distributed the same way as the output of the setup step for both H_{i-1} and H_i. Then, for the first $i-1$ queries, do what both H_{i-1} and H_i would do, namely, run Hard-Comm and Soft-Comm algorithms in response to the corresponding queries. For query i, do not run any algorithm, just output $C = (C_1, C_2)$. If ever queried for Tease or Open corresponding to these, run Sim-Tease or Sim-Open. For queries beginning with query $i+1$, run Sim-Commit. It is clear that, by using the distinguisher, one can determine whether or not (C_1, C_2) are commitments to $(0, 0)$.

□

Remark. As noted in the introduction, the same construction can be used to achieve interactive mercurial commitments, from standard commitments and (interactive) zero knowledge proofs. Since both these building blocks are implied by one-way functions, the construction yields interactive mercurial commitments based on one-way functions. Noninteractive mercurial commitments based on one-way-functions are constructed in [CDV06, Section 4].

2.3.2 From Claw-Free Trapdoor Bijections

We now construct a mercurial bit-commitment scheme under the assumption that there exist *claw-free trapdoor bijections.*[4] Specifically, slightly generalizing the notion of claw-free permutations of [GMR88], we assume that there exist indexed families of bijections $\{f_i\}_{i \in I \subseteq \{0,1\}^n}$, $f_i : D_{f_i} \to R_i$ and $\{g_i\}_{i \in I \subseteq \{0,1\}^n}$, $g_i : D_{g_i} \to R_i$, and an efficiently computable distribution Δ on pairs $(i, t_i) \in \{0,1\}^n \times \{0,1\}^{\mathrm{poly}(n)}$ such that:

- t_i is trapdoor information that allows f_i to be inverted efficiently.
- f_i and g_i are claw-free. That is, when given i sampled according to Δ, no efficient algorithm can, with nonnegligible probability, find $s \in D_{f_i}$ and $s' \in D_{g_i}$ such that $f_i(s) = g_i(s')$.

Employing this assumption, we construct mercurial bit-commitments:

- $PK = \text{Setup}(1^n) = i$ where (i, t_i) is sampled from Δ
- Using randomness $(r_0, r_1) \in D_{f_i} \times D_{g_i}$,

$$\text{Hard-Comm}(i, 0, (r_0, r_1)) = (f_i(r_0), g_i(r_1)) \quad \text{and} \quad \text{Hard-Comm}(i, 1, (r_0, r_1)) = (g_i(r_1), f_i(r_0))$$

- Using randomness $(r_0, r_1) \in D_{f_i} \times D_{f_i}$, $\text{Soft-Comm}(i, (r_0, r_1)) = (f_i(r_0), f_i(r_1))$
- For a hard commitment $C = (C_0, C_1)$, $\tau = \text{Tease}(i, x, (r_0, r_1), (C_0, C_1)) = r_0$.
- For a soft commitment $C = (C_0, C_1)$, $\tau = \text{Tease}(i, x, (r_0, r_1), (C_0, C_1)) = r_x$.
- For both hard and soft commitments $\text{Ver-Tease}(i, x, \tau, (C_0, C_1))$ checks that $C_x = f_i(\tau)$.
- To open a hard commitment $C = (C_0, C_1)$ to x, created using the random string (r_0, r_1), $\pi = \text{Open}(i, x, (r_0, r_1), (C_0, C_1)) = (x, r_0, r_1)$.
- Given a decommitment $\pi = (x, r_0, r_1)$, $\text{Ver-Open}(i, x, \pi, (C_0, C_1))$ checks $C_x = f_i(r_0)$ and $C_{1-x} = g_i(r_1)$.

[4] Note that in contrast to the construction of the previous section, here we construct a bit-commitment scheme, i.e. we commit only to values $x \in \{0, 1\}$.

The correctness of this commitment scheme is immediate from the above descriptions. Furthermore, it is clear that these commitments are hiding since all commitments are pairs of completely random elements of R_i. That hard commitments are binding follows from the assumption that f_i and g_i are claw-free.

It remains to show that this commitment scheme is simulatable. The key step in showing simulatability is to note that if t_i (i.e. the trapdoor for f_i) is known, then one can easily compute $f_i^{-1}(s)$ for any given $s \in R_i$, and in particular, one can produce an r' such that $s = f_i(r')$, even if s was chosen to be $g_i(r)$ for some random $r \leftarrow D_{g_i}$. Thus, if one knows t_i, then any "commitment" C of the form $(C_0, C_1) = (g_i(r_0), g_i(r_1))$ can be opened arbitrarily. More formally, we have the following simulator.

- $\textsc{Sim-Setup}(1^n)$ outputs the pair (i, t_i).
- All commitments $C = (C_0, C_1)$ are generated as $\textsc{Sim-Commit}((i, t_i), (r_0, r_1)) = (g_i(r_0), g_i(r_1))$ using randomness $(r_0, r_1) \leftarrow D_{g_i} \times D_{g_i}$.
- The teaser value is computed as $\textsc{Sim-Tease}((i, t_i), x, (r_0, r_1), (C_0, C_1)) = f_i^{-1}(C_x)$.
- Finally, the simulator can open $C = (C_0, C_1) = (g_i(r_0), g_i(r_1))$ arbitrarily using

$$\textsc{Sim-Open}((i, t_i), 0, (r_0, r_1)) = (x, f_i^{-1}(C_0), r_1) \quad \text{and} \quad \textsc{Sim-Open}((i, t_i), 1, (r_0, r_1)) = (x, r_0, f_i^{-1}(C_1)) \,.$$

Still, all commitments are random elements of $R_i \times R_i$, and since f_i and g_i are permutations, all decommitments are unique, so the outputs of the simulator are distributed identically to the outputs of a true committer. Thus, the above mercurial bit-commitment scheme is simulatable; moreover, the simulation is perfect.

Claw-free trapdoor bijections are an established cryptographic primitive [GMR88]. They are commonly realized under the assumption that factoring is hard. However, under the factoring assumption one can construct much more efficient mercurial commitments, as we do later in this section. Nonetheless, the above construction based on a claw-free pair is valuable because the existence of a claw-free pair may be viewed as a generic assumption independent of the difficulty of factoring. Indeed, it seems reasonable that there should be many different pairs of functions satisfying this assumption. Note that only f_i requires a trapdoor, so g_i may be some, completely unrelated, one-way bijection. It may be reasonable to assume that finding a claw is infeasible when f_i and g_i have very little to do with each other.

2.3.3 From the Discrete Logarithm Assumption

The following mercurial commitment scheme relies on the intractability of the discrete logarithm problem in a group G of prime order. When G is taken to be the subgroup of size q of $\mathbb{Z}_p^*$ where $q|(p-1)$ (i.e., G is the group of d-th order residues in $\mathbb{Z}_p^*$ for a prime $p = dq + 1$), this mercurial commitment scheme is implicit in the Zero-Knowledge Sets construction of [MRK03]. Indeed, combining this mercurial commitment with the Zero-Knowledge Set construction of the next section yields essentially the same construction as [MRK03].

This mercurial commitment scheme is in the shared random string model.[5]

Recall that the Pedersen commitment scheme [Ped92] employs two randomly chosen generators, $g, h \leftarrow G$, and a commitment to a message $m \in \{0, 1, \ldots, |G| - 1\}$ is computed by selecting a random $r \leftarrow \{0, 1, \ldots, |G| - 1\}$ and letting the commitment be $g^m h^r$. The commitment is opened by revealing the

[5]Actually, this scheme in the shared random string model only if a description of G that allows for efficient group operation, as well as two elements g, h such that $\log_g(h)$ is hard to compute, can be generated from a shared random string. This is the case for all groups commonly used in discrete-logarithm-based schemes, such as $\mathbb{Z}_p^*$ and its subgroups, as well as elliptic curve groups. It is possible, however, that there are also groups where DL is hard only if the group description is generated using some secret coins.

message m and the random exponent r. It is not hard to show that if the committer can open this commitment in more than one way, then he can easily compute $\log_g(h)$, a task which is presumed to be intractable. On the other hand, if the committer knows $\log_g(h)$, then he can easily open a supposed commitment $c = g^k \in G$ to any message m by producing the pair $(m, (k-m)/\log_g(h) \bmod |G|)$. This observation is essential to the following mercurial commitment scheme which is based on the Pedersen commitment.

- $\textsc{Setup}(1^k)$ selects prime order group G and generators $(g,h) \leftarrow G \times G$ (with the requirement that $g, h \neq 1$). Let the message space be $\{0, \ldots, |G|-1\}$.
- The hard commitment to a value x (with random strings $r_0 \leftarrow \{0,1,\ldots,|G|-1\}$ and $r_1 \leftarrow \{1,\ldots,|G|-1\}$) is simply the Pedersen commitment to x using the generator pair (g, h^{r_1}): $\textsc{Hard-Comm}((g,h), x, (r_0, r_1)) = (g^x (h^{r_1})^{r_0}, h^{r_1})$.
- The soft commitment (with $r_0 \leftarrow \{0,1,\ldots,|G|-1\}$ and $r_1 \leftarrow \{1,\ldots,|G|-1\}$) is: $\textsc{Soft-Comm}((g,h),(r_0,r_1)) = (g^{r_0}, g^{r_1})$.
- For a hard commitment $C = (C_0, C_1)$, $\tau = \textsc{Tease}((g,h), x, (r_0,r_1), (C_0,C_1)) = r_0$.
- For a soft commitment $C = (C_0, C_1)$, $\tau = \textsc{Tease}((g,h), x, (r_1,r_2), (C_0,C_1)) = (r_0 - x)/r_1 \bmod |G|$.
- In either case, $\textsc{Ver-Tease}((g,h),(C_0,C_1),x,\tau)$ checks that $C_0 = g^x \cdot C_1^{\tau}$.
- $\textsc{Open}$ computes (π_0, π_1) as $\textsc{Open}((g,h), x, (r_0,r_1),(C_0,C_1)) = (r_0, r_1)$.
- Finally, verification is similar to Pedersen's, with an additional step to ensure that the second generator C_1 was chosen as a known power of h, and hence that $\log_g(C_1)$ is not known to the committer: $\textsc{Ver-Open}((g,h),(C_0,C_1),x,(\pi_0,\pi_1))$ checks that $C_0 = g^x \cdot C_1^{\pi_0}$ and that $C_1 = h^{\pi_1}$.

The correctness of the above algorithms is easily verified. The proof that hard commitments are binding is just as with the Pedersen commitment; indeed, the ability to open a commitment $C = (C_0, C_1)$ in two ways implies knowledge of $\log_g(h)$. This scheme is clearly hiding because all commitments consist of random elements from $G \times G$. As for simulatability, we define the simulator as follows.

- Let $\textsc{Sim-Setup}(1^k) = ((g,h), TK)$ where $g \leftarrow (G \setminus \{1\})$, $TK \leftarrow \{1,\ldots,|G|-1\}$ and $h = g^{TK}$. Observe that now $\log_g(h) = TK$ is known to the simulator. It is not hard to see that all the above commitments can be opened arbitrarily when $\log_g(h)$ is known, and thus the simulator can easily open any commitments that it needs to, as follows.
- $\textsc{Sim-Commit}((PK, TK),(r_0,r_1)) = (g^{r_0}, g^{r_1})$ (where $r_0 \leftarrow \{0,1,\ldots,|G|-1\}$ and $r_1 \leftarrow \{1,\ldots,|G|-1\}$).
- $\textsc{Sim-Tease}((PK, TK), r, x) = (r_0 - x)/r_1 \bmod |G|$.
- $\textsc{Sim-Open}((PK, TK),(r_0,r_1),x) = ((r_0 - x)/r_1 \bmod |G|, r_1/TK \bmod |G|)$.

Finally, we note that the distribution of outputs from these simulator algorithms is identical to the distribution of outputs from a true committer: in both cases, g and h are random generators of G and all commitments are random pairs $(C_0, C_1) \leftarrow G \times (G \setminus \{1\})$. Thus, this mercurial commitment scheme is simulatable; moreover, the simulation is perfect.

2.3.4 From the Hardness of Factoring

Our final construction is based on the hardness of factoring. Like the discrete logarithm construction, this scheme commits to many bits simultaneously. This is a modification of the trapdoor commitment construction implicit in the GMR signature scheme [GMR88]. We note that a similar mercurial commitment scheme (based on RSA rather than factoring, but allowing for interesting extensions to nonmalleability based on the Strong RSA assumption) was independently discovered by Gennaro and Micali [GM06].

The commitment scheme from [GMR88] commits to an ℓ-bit message m by publishing $t^m r^{2^\ell}$, for some fixed known $t \in QR_N$ and random $r \in \mathbb{Z}_N^*$. Decommitment involves simply revealing r. The ability to break binding and decommit to two different values implies the knowledge of a square root of t; conversely, if one knows 2^ℓ-th root of t, then one can decommit to any value. Thus, the main idea for turning this into mercurial commitments is to vary t, using one whose 2^ℓ-th root is known for soft commitments, and one who square root is provably unknown for the hard commitments. (Note that this approach is similar in spirit to the discrete-logarithm-based scheme of Section 2.3.3, where a different generator is used each time, whose discrete logarithm is known for soft commitments and unknown for hard commitments.)

We first consider the case where $N = pq$ such that p, q are safe primes (i.e. there exist primes p', q' such that $p = 2p' + 1$ and $q = 2q' + 1$), as this leads to a simpler construction and simpler analysis. Then we describe a more general scheme which we show is secure for any RSA modulus N.

A simple scheme The mercurial commitment scheme runs as follows:

- Let the message space be $\{0, 1\}^\ell$.
- $\textsc{Setup}(1^n)$ chooses an RSA modulus $N = pq$, where p, q are safe primes, and a random element $y \in \mathbb{Z}_N^*$. Let $U = y^2$. $PK = (N, U)$.
- Using randomness $(r_0, r_1) \in \mathbb{Z}_N^* \times \mathbb{Z}_N^*$, $\textsc{Hard-Comm}((N, U), m, (r_0, r_1)) = (r_0^{2^\ell}(Ur_1^{2^\ell})^m, Ur_1^{2^\ell})$.
- Using randomness $(r_0, r_1) \in \mathbb{Z}_N^* \times \mathbb{Z}_N^*$, $\textsc{Soft-Comm}((N, U), (r_0, r_1)) = (r_0^{2^\ell}, r_1^{2^\ell})$.
- For a hard commitment $C = (C_0, C_1)$, $\tau = \textsc{Tease}((N, U), m, (r_0, r_1), (C_0, C_1)) = r_0$.
- For a soft commitment $C = (C_0, C_1)$, $\tau = \textsc{Tease}((N, U), m, (r_0, r_1), (C_0, C_1)) = r_0 r_1^{-m}$.
- $\textsc{Ver-Tease}((N, U), m, \tau, (C_0, C_1))$ checks that $C_0 = C_1^m (\tau)^{2^\ell}$.
- To open a hard commitment $C = (C_0, C_1)$ to m, created using the random string (r_0, r_1), $\pi = \textsc{Open}((N, U), m, (r_0, r_1), (C_0, C_1)) = (m, r_0, r_1)$.
- Given a decommitment $\pi = (m, r_0, r_1)$, $\textsc{Ver-Open}((N, U), m, \pi, (C_0, C_1))$ checks $C_1 = Ur_1^{2^\ell}$ and $C_0 = C_1^m r_0^{2^\ell}$.

The correctness of this commitment scheme follows directly from the above definitions. Simulatability is also fairly simple:

- $\textsc{Sim-Setup}(1^n) = (N, U, y)$ where N is generated as before, y is a random element from $\mathbb{Z}_N^*$, and $U = y^{2^\ell}$. Because squaring is a permutation over quadratic residues (for $N = pq$ such that $p \equiv q \equiv 3 \pmod 4$), U is still a random square.
- $\textsc{Sim-Commit}((N, U, y), (r_0, r_1)) = (r_0^{2^\ell}, Ur_1^{2^\ell})$.
- $\textsc{Sim-Tease}((N, U, y), m, (r_0, r_1), (C_0, C_1)) = r_0(yr_1)^{-m}$.

- Sim-Open$((N, U, y), m, (r_0, r_1)) = (r_1, r_0(yr_1)^{-m})$.

Again the output of this simulator and of the original scheme are distributed identically. Sim-Setup as argued above produces the same distribution on the parameters as Setup. In both the real and the simulated games, every commitment is a pair of random squares. The opening of a valid commitment (C_0, C_1) to a message m will consist of two values; in both games, the first will be uniformly distributed over the 2^ℓth roots of C_0/C_1^m, and the second will be uniformly distributed over the 2^ℓth roots of C_1/U. Similarly, the outputs of Tease and Sim-Tease will be identically distributed. Thus the commitment scheme is perfectly simulatable.

Now we have only to show that this scheme is binding. Suppose there exists a hard commitment (C_0, C_1) which can be opened as (m, r_0, r_1), and (m', r'_0, r'_1), where $m = b_1 \dots b_\ell$, and $m' = b'_1 \dots b'_\ell$. Both openings can be successfully verified, thus we have $C_1 = Ur_1^{2^\ell} = Ur_1'^{2^\ell}$, and $C_0 = C_1^m r_0^{2^\ell} = C_1^{m'} r_0'^{2^\ell}$. Given that $m \neq m'$ and p, q are safe primes, this means that either we trivially factor N (by finding p' or q'), or $r_0 \neq r'_0$. Consider the second case and let $f_0(x) = x^2$, $f_1(x) = C_1 x^2$. Note that finding a claw (i.e. x_0, x_1 such that $f_0(x_0) = f_1(x_1)$) would give a square root of U: $(U = (x_0 x_1^{-1} r_1^{-2^{\ell-1}})^2)$. This would then allow us to factor N with nonnegligible probability. Thus, this is a claw-free pair. Note also that $C_1^m r_0^{2^\ell} = f_{b_\ell}(f_{b_{\ell-1}}(\dots(f_{b_1}(r_0))))$. Since there are two verifiable openings, this must be equal to $C_1^{m'} r_0'^{2^\ell} = f_{b'_\ell}(f_{b'_{\ell-1}}(\dots(f_{b'_1}(r'_0))))$. Let i be the smallest index such that $f_{b_i}(f_{b_{i-1}}(\dots(f_{b_1}(r_0)))) = f_{b'_i}(f_{b'_{i-1}}(\dots(f_{b'_1}(r'_0))))$. Such an i must clearly exist, and as long as $r'_0 \neq r_0$ we also have $b_i \neq b'_i$. Thus we have found a claw between f_0 and f_1 which will allow us to factor N with nonnegligible probability.

A similar proof shows that we cannot tease-open to one value and hard open to another.

A scheme with arbitrary RSA modulus We will now show that a similar scheme using an arbitrary RSA modulus N can be created using a modified version of the commitment described in [Fis01]as follows:

- Let $\ell' = \ell + n$ where ℓ is the message length as above.
- Setup(1^n) chooses an odd n-bit RSA modulus $N = pq$, and a random element $y \in \mathbb{Z}_N^*$. Let $U = y^{2^n}$. $PK = (N, U)$.
- All other algorithms proceed as in the simple scheme, using ℓ' in place of ℓ.

Again, correctness follows directly from these definitions. Simulatability is obtained by choosing $U = y^{2^{\ell'}}$ for random y. For an odd n bit modulus, squaring is a permutation on the 2^nth powers. Since $\ell' > n$ this means U is distributed like the 2^nth powers as in the Setup above. The rest of Simulatability follows from the same arguments as for the simple scheme.

The binding property is a bit more complicated. Here we will use a result by Fischlin and Fischlin [FF02]. Fischlin and Fischlin define the factoring representation problem, in which (informally) the adversary is given a random n-bit RSA modulus N, an integer ℓ, and a random element $g \leftarrow \{h^{2^n} | h \in Z_N^*\}$, and is asked to produce $(x_1, y_1), (x_2, y_2) \in Z_{2^\ell} \times Z_N^*$ such that $x_1 \neq x_2$ and $g^{x_1} y_1^{2^{n+\ell}} = g^{x_2} y_2^{2^{n+\ell}}$.[6] They show that the factoring representation problem is equivalent to the factoring problem. Thus, we have only to show that breaking the binding property of our mercurial commitment scheme will imply a solution to the factoring representation problem.

Suppose we have an adversary who, when given N, ℓ, U can with nonnegligible produce a hard opening (m, r_0, r_1) and a tease opening (m', r'_0) for a commitment (C_0, C_1) with $m \neq m'$. Then if the verification algorithms accept, we have $C_1 = Ur_1^{2^{\ell'}}$ and $C_0 = r_0^{2^{\ell'}} C_1^m = {r'_0}^{2^{\ell'}} C_1^{m'}$. A little arithmetic gives us $U^m (r_1^m r_0)^{2^{\ell'}} = U^{m'} (r_1^{m'} r'_0)^{2^{\ell'}}$. Thus, $m, m', (r_1^m r_0), (r_1^{m'} r'_0)$ are a solution to the representation problem.

[6] Here we have changed the notation and simplified the definition of the problem slightly to match our application.

Similarly, we can show that we cannot hard open a commitment to two different messages. Thus, the binding property holds.

3 Constructing Zero-Knowledge Sets

3.1 Definition of Zero-Knowledge Sets

Let us start with an informal definition. A zero-knowledge set scheme (more generally, an elementary database scheme) [MRK03] consists of three algorithms, ZKS-Setup, P (the Prover) and V (the Verifier) such that three properties hold: (1) completeness: for any database D, for any x such that $D(x) = v$ (where v can be a value if $x \in D$ or $\perp$ if $x \notin D$) an honest Prover who correctly commits to D can always convince the verifier that $D(x) = v$; (2) soundness: once a commitment to the database D has been formed (even by a malicious Prover), no P' can, for the same x, convince the Verifier that $D(x) = v_1$ and $D(x) = v_2$ for $v_1 \neq v_2$; (3) zero-knowledge: there exists a simulator S such that even for adversarially chosen D, no adversarial verifier can tell whether he is (a) talking to an honest prover P committed to D, or (b) talking to S who only has oracle access to the data set D.

Let us now give the formal definition. This definition is a revision of the original one due to [MRK03]; we explain the differences below.

Definition 3.1. A database D is a map $\{0,1\}^* \to \{0,1\}^* \cup \{\perp\}$ such that $D(x) \neq \perp$ for only finitely many values x. The domain of D is called *keys* and the range is called *values*. We will say a key $x \in D$ if $D(x) \neq \perp$.

A ZKS consists of the probabilistic polynomial-time algorithms ZKS-Setup, $P = (P_1, P_2)$, and V, satisfying the following properties:

1. *Completeness.* For all databases D and for all x

$$\Pr[\sigma \leftarrow \text{ZKS-Setup}(1^k)\ ;\ (com, \mathtt{state}) \leftarrow P_1(1^k, \sigma, D)\ ;\ \pi_x \leftarrow P_2(\mathtt{state}, x) : \\ V(1^k, \sigma, com, x, D(x), \pi_x) = ACCEPT] = 1$$

2. *Soundness.* For all x and for all probabilistic polynomial-time malicious provers P',

$$\Pr[\sigma \leftarrow \text{ZKS-Setup}(1^k)\ ;\ (com, v_1, v_2, \pi_1, \pi_2) \leftarrow P'(1^k, \sigma) : \\ v_1 \neq v_2 \wedge V(1^k, \sigma, com, x, v_1, \pi_1) = ACCEPT \wedge V(1^k, \sigma, com, x, v_2, \pi_2) = ACCEPT]$$

is negligible in k (note that v_1 and v_2 can be strings or $\perp$).

3. *Zero-Knowledge.* There exists a simulator $Sim = (Sim_1, Sim_2, Sim_3)$ such that for probabilistic polynomial-time malicious verifiers $Adv = (Adv_1, Adv_2)$, the absolute value of the difference

$$\Pr[\sigma \leftarrow \text{ZKS-Setup}(1^k)\ ;\ (D, \mathtt{state}_A) \leftarrow Adv_1(1^k, \sigma)\ ; \\ (com, \mathtt{state}_P) \leftarrow P_1(1^k, \sigma, D) : Adv_2^{P_2(\mathtt{state}_P, \cdot)}(\mathtt{state}_A) = 1] - \\ \Pr[(\sigma, \mathtt{state}_\sigma) \leftarrow Sim_1(1^k)\ ;\ (D, \mathtt{state}_A) \leftarrow Adv_1(1^k, \sigma)\ ; \\ (com, \mathtt{state}_S) \leftarrow Sim_2(1^k, \mathtt{state}_\sigma) : Adv_2^{Sim_3(\mathtt{state}_S, \cdot, D(\cdot))}(\mathtt{state}_A) = 1]$$

is negligible in k. (Immediately above, the notation $Adv_1^{Sim_3(\mathtt{state}_S, \cdot, D(\cdot))}$ means that the adversary gets to choose x and will receive the result of $Sim_3(\mathtt{state}_S, x, D(x))$ — i.e., the two $\cdot$ symbols refer to the same value). This defines computational zero-knowledge; perfect and statistical zero-knowledge can be defined similarly, as discussed below.

There are two differences between this definition and the one due to Micali et al. First, the definition of [MRK03] is in the shared random string model: σ is simply a uniformly chosen random string, unless it is being produced by the simulator. We allow, more generally, for shared parameters to be generated by a (trusted) parameter generation algorithm. If this algorithm simply outputs its random coins, this puts us back in the shared random string model. Otherwise, we are in the trusted parameters model (which, of course, is a stronger assumption). The model of our construction depends on the model for the mercurial commitment on which it is based. As mentioned in the previous section, we have mercurial commitment constructions in both models.

Second, our definition of zero-knowledge holds not for all databases D, but only for those that Adv can output. In contrast, the definition of [MRK03] is stated for all D. This creates a problem: the "for all" quantifier makes it impossible to construct computational ZKS, unless it is also statistical ZKS,[7] because D may contain information that is not efficiently computable and is helpful to Adv. For example, the database D may include information for breaking the computational assumption used—e.g., information needed to invert the one-way function used in the assumption. More generally, $D(x)$ may simply interpret x as a bit string representing the adversary's view, and be equal to 0 if x more likely to be real and to 1 if x is more likely to be simulated. Since the adversary has oracle access to the true database (whether it is interacting with the true prover or the simulator), the adversary can query D on its view, and thus distinguish the true prover and the simulator. To make computational ZKS meaningful, therefore, one must confine oneself to databases that do not provide the adversary with powers otherwise unavailable to it. This is why we restrict the definition to polynomial-time computable databases.

3.2 ZKS from mercurial Commitments

In this section we show how, given a mercurial commitment scheme and a collision-resistant hash function, we can construct a zero-knowledge set. As already mentioned, this construction is essentially the same as the construction of [MRK03] with the mercurial commitments abstracted as a building block.

3.2.1 On the Role of Collision-Resistant Hashing

In order to construct ZKS from mercurial commitments, we need an additional property: that an ordered pair of commitments produced by $\textsc{Hard-Comm}(PK, \cdot, \cdot)$ and/or $\textsc{Soft-Comm}(PK, \cdot)$ is in the domain D_{PK} of the commitment scheme (this property is needed because we will build trees of commitments, with the parent committing to its two children). This can be accomplished for any mercurial commitment scheme with sufficiently large D_{PK} by combining it with a collision-resistant hash function H that maps pairs of commitments into D_{PK}. Then, to commit to a pair of commitments, one would simply commit to its hash value instead. This approach was already used by [MRK03] with the DL-based mercurial commitment scheme implicitly constructed there.

The key for the hash function can be included as part of the commitment scheme's public key. The security of the resulting construction is easy to prove (we will not do so here because the arguments are standard). From now on, in describing the ZKS construction from mercurial commitments, we will assume that domain of a mercurial commitment scheme includes pairs of commitments.

We note that it is *necessary* to assume collision-resistant hash functions (CRHFs) because they are implied by ZKS: to build CRHF from ZKS, we can view an arbitrary-length string $b_1 b_2 \ldots b_\ell$ as an elementary database D where $D(i) = b_i$ for $1 \leq i \leq \ell$, and $i > \ell$ is not in the database. To compute the hash function, simply run the prover algorithm on the database D to produce a commitment C, fixing the

[7] The ZKS *construction* of [MRK03] achieves perfect zero-knowledge, and so is not affected by this flaw in the definition. The *definition* of [MRK03], however, provides also for computational and statistical zero-knowledge; however, here we argue that their notion of computational zero-knowledge is not meaningful unless properly modified

prover-randomness to an all-0 string. It is easy to see that the resulting algorithm is collision-resistant: an adversary who could produce two strings (databases) that hash to the same commitment C would contradicts the soundness property of ZKS.

3.2.2 Building ZKS

Below, we show how to construct ZKS for a database D with keys of length[8] l given

- a mercurial commitment scheme (Setup, Hard-Comm, Soft-Comm, Tease, Open, Ver-Tease, Ver-Open) whose domain includes the values v contained in the database, as well as pairs of commitments (produced by Hard-Comm and/or Soft-Comm);
- a pseudorandom function family $\{F_s\}_{s \in S}$ that maps binary string of length up to l to binary strings of length needed for random inputs r to Hard-Comm and Soft-Comm.[9]

Our construction will be in the shared random string model if the mercurial commitment scheme (and the collision-resistant hash function, if separate from the mercurial commitment scheme) both require no more than a shared random string for their parameters. Otherwise, it will be in the trusted parameters model.

Intuition Informally, to generate a commitment *com* to the database D, the prover views each key x as an integer numbering a leaf of a height-l binary tree, and places a commitment to the information $v = D(x)$ into leaf number x. Each internal node of the tree is generated to contain the commitment to the contents of its two children. The result is thus a Merkle tree, except that each internal node is a commitment to, rather than a hash of, its two children. The value *com* is then the value contained in the root of the tree.

To respond to a query about x, the prover simply decommits the corresponding leaf and provides the authenticating path (together with all the decommitments) to the root.

The only problem with the just-described approach is that it requires exponential time (in l) to compute the tree, because the tree has 2^l leaves. This is where mercurial commitments help. Our exponential-size Merkle-like tree will have large subtrees where every leaf is a commitment to $\perp$, because the corresponding keys are not in the database. Instead of actually computing such a subtree ahead of time, the prover simply forms the root of this subtree as a soft-commitment, and does not do anything for other nodes of the subtree. Thus, the resulting Merkle tree gets "pruned" to size at most $2l|D|$, because the only nodes in the tree are ancestors of leaves in D and their siblings. (This is because if a node is neither an ancestor of a leaf in D nor a sibling of such an ancestor, then both its and its sibling's subtrees are empty.)

Answering queries about $x \in D$ is still done the same way. In response to queries about $x \notin D$ the prover generates the portion of the subtree that is missing (from x to the root of the empty subtree). The value at the root of the empty subtree is then teased (soft-decommitted) to its two (newly generated) children, and the entire authenticating path from x to *com* is provided using teasers, rather than hard decommitments. This is illustrated in Figure 1.

To save the prover from the expense of having to remember all the choices it made when generating the tree (both initially and in response to $x \notin D$ queries), the prover can generate all random strings used in the commitments pseudorandomly rather than randomly.

[8] As suggested in [MRK03], we can apply collision-resistant hashing to the keys if they are longer, although we will give up perfect completeness if two keys x_1 and x_2 such $D(x_1) \neq D(x_2)$ hash to the same result.

[9] The pseudorandom function family is needed only to save prover storage and make the prover stateless; if the prover is willing to store the amount of randomness proportional to $l(|D| + q)$, where q is the number of queries, then it is not necessary.

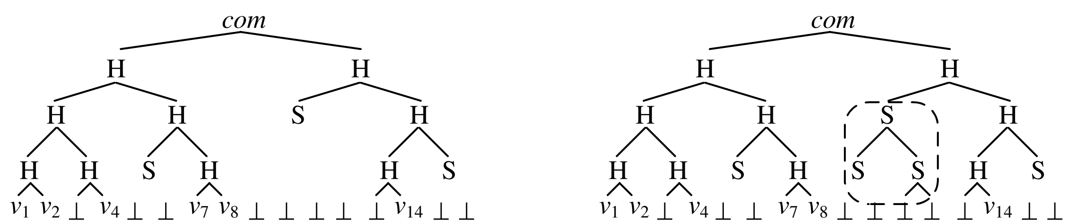

Figure 1: A commitment tree before and after a query for key 11, whose value is not in the database. The parts built in response to the query are shown in the second tree. Hard commitments are denoted by 'H' and soft commitments by 'S'. Each leaf contains a commitment to the value shown rather than the value itself.

Soundness follows from the fact that soft decommitments always have the same semantics, namely that $x \notin D$, and that soft decommitments cannot, by the binding property, disagree with hard decommitments. Zero-knowledgeness follows from the simulatability of commitments and from the fact that decommitments are consistent: a given node will never be (hard- or soft-) decommitted in two different ways. Note that zero-knowledge will be perfect, statistical, or computational, depending on the simulatability of mercurial commitments (however, for perfect and statistical zero-knowledge, the prover must use truly random, rather than pseudorandom, strings; hence, it must be stateful in order to remember the random strings it used when responding to queries.)

We formalize the above description in the following section. There are no surprises there, and a reader able to fill in the details given the intuitive description above can safely skip it.

3.3 Detailed Description of ZKS from Mercurial Commitments

Here we describe the precise details of the construction of Zero-Knowledge sets from mercurial commitments.

- **Committing to the Database.** To generate the commitment *com* to the database D, the prover generates an (incomplete) binary tree of commitments—resembling a Merkle tree—as follows. First, choose a pseudorandom function F_s from the family by randomly choosing a seed s. To simply notation, we will denote $F_s(a)$ by r_a in the sequel. For each x such that $D(x) = v \neq \perp$, produce $C_x = \textsc{Hard-Comm}(PK, v, r_x)$. Now, for each x such that $D(x) = \perp$ but $D(x') \neq \perp$, where x' denotes x with the last bit flipped, produce $C_x = \textsc{Soft-Comm}(PK, r_x)$. Define $C_x = nil$ for all other x. Now build the tree in a bottom-up fashion as follows: for each level i from $l-1$ to 0, and for each string σ of length i, define C_σ as follows:

  1. For all σ such that $C_{\sigma 0} \neq nil$ and $C_{\sigma 1} \neq nil$, let $C_\sigma = \textsc{Hard-Comm}(PK, (C_{\sigma 0}, C_{\sigma 1}), r_\sigma)$.
  2. For all σ such that $C_{\sigma'}$ has been defined in step 1 but C_σ has not been (where σ' denotes σ with the last bit flipped), define $C_\sigma = \textsc{Soft-Comm}(PK, r_\sigma)$.
  3. For all other σ, define $C_\sigma = nil$

  If the value at the root $C_\epsilon = nil$ (this will happen only if $D = \emptyset$), redefine $C_\epsilon = \textsc{Soft-Comm}(PK, r_\epsilon)$. Finally, *com* is defined to be C_ϵ.

- **Answering Queries.** When the prover receives a query about a value x that is in the database, proceed as follows. Let $x|_i$ be the first i bits of x, and $(x|_i)'$ be the first $i-1$ bits of x followed by the i-th bit flipped. Let $\pi_x = \textsc{Open}(PK, D(x), r_x, C_x)$ and $\pi_{x|_i} = \textsc{Open}(PK, (C_{x|_i 0}, C_{x|_i 1}), r_{x|_i}, C_{x|_i})$ for $0 \leq i < l$. Return $D(x)$ together with $C_{x|_i}$, $C_{(x|_i)'}$ for $1 \leq i \leq l$ and $\pi_{x|_i}$ for $0 \leq i \leq l$. In other words, the prover returns $D(x)$ together with its authenticating path to the root, which consists of ancestors of x, their siblings, and proofs that each parent is the commitment to the two children.

If the prover receives a query about a value x that is not in the database, proceed as follows. If $C_x = nil$, let h be largest value such that $C_{x|_h} \neq nil$, let $C_x = \text{Soft-Comm}(PK, r_x)$, and build a path from x to $C_{x|_h}$ as follows: let $C_{x'} = \text{Soft-Comm}(PK, r_{x'})$; for each level i from $l-1$ to $h+1$, define $C_{x|_i} = \text{Soft-Comm}(PK, r_{x|_i})$, and $C_{(x|_i)'} = \text{Soft-Comm}(PK, r_{(x|_i)'})$. Note that the only values inside the tree redefined by the above procedure are those that were *nil* before. Let $\tau_x = \text{Tease}(PK, \perp, r_x, C_x)$ and $\tau_{x|_i} = \text{Tease}(PK, (C_{x|_i0}, C_{(x|_i1)'}), r_{x|_i}, C_{x|_i})$ for $0 \le i < l$. Return $\perp$ together with $C_{x|_i}$, $C_{(x|_i)'}$ for $1 \le i \le l$ and $\pi_{x|_i}$ for $0 \le i \le l$. In other words, the prover returns $\perp$ together with its authenticating path to the root, which consists of ancestors of x, their siblings, and teaser-proofs that each parent is the commitment to the two children.

- **Verifying Answers.** If the answer is not $\perp$, the verifier does the usual Merkle tree verification, except with commitments instead of hash functions: he simply performs $\text{Ver-Open}(PK, C_{x|_i}, (C_{x|_i0}, C_{x|_i1}), \pi_x)$ for $1 \le i < l$, as well as $\text{Ver-Open}(PK, com, (C_0, C_1), \pi_\epsilon)$ and $\text{Ver-Open}(PK, C_x, D(x), \pi_x)$. If the answer is $\perp$, the verifier also does the usual Merkle tree verification, except with teaser-proofs instead of hash functions: he simply performs $\text{Ver-Tease}(PK, C_{x|_i}, (C_{x|_i0}, C_{(x|_i1)}), \pi_x)$ for $1 \le i < l$, as well as $\text{Ver-Tease}(PK, com, (C_0, C_1), \tau_\epsilon)$ and $\text{Ver-Tease}(PK, C_x, \perp, \tau_x)$.

Security Here we provide only a sketch of the security proof.

The scheme described above is complete, because hard commitments can always be (soft- or hard-) decommitted to the values they were committed to, and soft commitments can always be soft-decommitted to any value.

Soundness follows by reduction to the binding property of the commitments. Indeed, suppose that two different answers v_1 and v_2 to the same query x are produced by a malicious prover and accepted by the verifier. Consider the following two cases. If $v_1 \neq \perp$ and $v_2 \neq \perp$, then for some commitment on the path from x to the root, Ver-Open returns $ACCEPT$ for two different decommitted values. If $v_1 \neq \perp$ and $v_2 = \perp$, or $v_1 = \perp$ and $v_2 \neq \perp$, then for some commitment on the path from x to the root, Ver-Open and Ver-Tease return $ACCEPT$ for two different decommitted values. (In both cases, the same commitment and the two different decommitted values exist because the root is always the same, but the leaves are different.) Either case contradicts the binding property of the commitment scheme.

Finally, zero-knowledgeness can be proven as follows. First, by pseudorandomness of F, our prover is indistinguishable from a prover who uses truly random strings. The zero-knowledge simulator will run Sim-Setup to generate (PK, TK), and then use Sim-Commit to generate a "commitment" in the roof of the tree. In response queries, it will use Sim-Commit to generate not-yet-generated nodes on the path from the root to the queried leaf, and use Sim-Open or Sim-Tease (depending of whether $x \in D$ or not) to decommit nodes on the path to their children. The result will be indistinguishable (from the prover who uses truly random strings) simply by the simulatability of the mercurial commitment scheme: the malicious verifier and the distinguisher from the definition of ZKS can be jointly viewed as the machine $D_k^{O_b}$ from the definition of mercurial commitments, making queries to the simulator or to the real prover. Zero-knowledge will be perfect, statistical, or computational, depending on the simulatability of mercurial commitments (however, for perfect and statistical zero-knowledge, the prover must use truly random, rather than pseudorandom, strings; hence, it must be stateful in order to remember the random strings it used when responding to queries.)

3.3.1 Efficiency

In the above, the specific choice of mercurial commitment, collision-resistant hash function and pseudorandom function will affect the efficiency of the protocol. We discuss these efficiency considerations in more detail below.

Let us fix a security parameter k, and we shall assume that all database keys are hashed to k bits, so that $l = k$. Moreover, we assume that the collision-resistant hash is built into the mercurial commitment scheme, allowing us to form k-bit commitments to *pairs* of k-bit strings. In this case, the construction of the commitment to the database D clearly requires $O(l \cdot |D|) = O(k \cdot |D|)$ calls to the pseudorandom function and $O(l \cdot |D|) = O(k \cdot |D|)$ mercurial commitments.

Once the database has been committed, proofs of membership or nonmembership consist of l mercurial decommitments each. Thus, if our mercurial commitment scheme allows committing to $2k$-bit strings with decommitments of length $O(k)$ (as is true for our mercurial commitments based on the discrete logarithm or factoring assumptions), then such a proof has length $O(l \cdot k) = O(k^2)$. On the other hand, if we have only mercurial bit-commitment with decommitments of length $O(k)$ (as is the case for our mercurial commitments based on trapdoor claw-free permutations), then a decommitment to a k-bit string has length $O(k^2)$ and the resulting proof has length $O(l \cdot k^2) = O(k^3)$.

Finally, the verifier needs to verify only $O(l) = O(k)$ mercurial decommitments to accept the proof's validity.

3.3.2 Improving the Efficiency of DL-based construction

While in general any collision-resistant hash function can be used with our DL-based mercurial commitments, Pedersen's hash function [Ped92] is suggested by [MRK03] because it is also based on the discrete logarithm assumption and its parameters can be selected from a shared random string. Given a group G of prime order q and two generators g and h, Pedersen's hash function $H_{G,g,h}$ hashes two integers $0 \leq a, b < q$ into a single element $h \in G$ via $h = g^a h^b$. It is easy to see that this hash function is collision-resistant if discrete logarithms in G are hard.

It may seem that Pedersen's hash function is well suited for use with DL-based mercurial commitments over the same group G. This, however, is not true, because the range of hash function is G, while the domain of the commitments is $\{0, 1, \ldots, q-1\}$. In particular, if G is the subgroup of size q of $\mathbb{Z}_p^*$ for $q|(p-1)$, one would need to choose two separate sets of parameters: (q_1, p_1) for the hash function and (q_2, p_2) for the commitment scheme, with $q_2 \geq p_1$. This seems to be necessary for the construction of [MRK03] to work (although it is not explicitly stated there).

In addition, to hash two commitments (which consist of two elements of $\mathbb{Z}_{p_2}^*$ each), multiple iterations of the hash function are needed, because a single iteration can handle only a pair of elements of $\mathbb{Z}_{q_1}$.

Here, we point out two minor modifications of Pedersen's hash function that eliminate the need for a second set of parameters and minimize the number of iterations necessary to hash two commitments. Both modifications rely on folklore results.

First, we will modify the hash function to take four inputs instead of two by using four generators (all in the shared random string), g_1, g_2, g_3, g_4, and outputting, on input (a, b, c, d), the value $g_1^a g_2^b g_3^c g_4^d$. The proof of collision-resistance of this function is a simple exercise and is omitted here.

Our second modification relies (in addition to the DL assumption) on the assumption that Sophie Germain primes are sufficiently dense (recall that a q is a Sophie Germain prime if $p = 2q+1$ is prime). We let q be a Sophie Germain prime. Then the subgroup G of order q of $\mathbb{Z}_p^*$ is QR_p. Consider the following efficient bijection ϕ' between QR_p and $\{1, 2, \ldots, q\}$: if $x \leq q$, $\phi'(x) = x$; else $\phi'(x) = p - x$ (this is a bijection because exactly one of $(x, -x)$ is in QR_p, because $p \equiv 3 \pmod 4$ since q is odd). Now let $\phi(x) = \phi'(x) - 1$ to make the range of the bijection $\{0, 1, \ldots, q-1\}$.

The bijection ϕ allows us to view $G = QR_p$ and $\{0, 1, \ldots, q-1\}$ as essentially the same.[10] Thus, we will simply modify Pedersen's hash function to output $\phi(h)$ instead of h, and to take inputs in QR_p instead of $\{0, 1, \ldots, q-1\}$ by applying ϕ^{-1} to them first.

The resulting ZKS scheme takes seven values and seven exponentiations per level of the hash tree to verify (four for the hash function and three for the mercurial commitment). Note that two of the generators can be shared between the hash function and the mercurial commitment scheme. Because verifications require only products of fixed-base exponentiations with four bases (except in the case of tease verification, when a single exponentiation with a random base is required), much precomputation can be done to speed up verification (see, e.g., [LL94], which can be extended to multiple bases).

Acknowledgments

The authors wish to thank anonymous referees for a detailed reading of the paper and useful suggestions, Ivan Visconti and Yevgeniy Dodis for clarifying subsequent work, Rosario Gennaro and Adi Shamir for helpful technical discussions, and Monica Perez for giving our new commitments their name "mercurial."

The authors were supported in part by the following: Melissa Chase by NSF Grant CNS-0347661 and NSF Graduate Research Fellowship; Alexander Healy by NSF Grant CCR-0205423 and a Sandia Fellowship; Anna Lysyanskaya by NSF Grant CNS-0374661; Tal Malkin by NSF Grant CCF-0347839; and Leonid Reyzin by NSF Grants CNS-0202067, CCR-0311485, CNS-0546614, CNS-0831281, CNS-1012798, and CNS-1012910.

References

[BCC88] Gilles Brassard, David Chaum, and Claude Crépeau. Minimum disclosure proofs of knowledge. *Journal of Computer and System Sciences*, 37(2):156–189, Oct. 1988.

[BDMP91] Manuel Blum, Alfredo De Santis, Silvio Micali, and Guiseppe Persiano. Non-interactive zero-knowledge. *SIAM Journal of Computing*, 20(6):1084–1118, 1991.

[BY96] Mihir Bellare and Moti Yung. Certifying permutations: non-interactive zero-knowledge based on any trapdoor permutation. *Journal of Cryptology*, 9(3):149–166, 1996.

[CDV06] Dario Catalano, Yevgeniy Dodis, and Ivan Visconti. Mercurial commitments: Minimal assumptions and efficient constructions. In Shai Halevi and Tal Rabin, editors, *Third Theory of Cryptography Conference, TCC 2006*, volume 3876 of *Lecture Notes in Computer Science*, pages 120–144. Springer, 2006.

[CFM08] Dario Catalano, Dario Fiore, and Mariagrazia Messina. Zero-knowledge sets with short proofs. In Nigel P. Smart, editor, *Advances in Cryptology — EUROCRYPT 2008*, volume 4965 of *Lecture Notes in Computer Science*, pages 433–450. Springer, 2008.

[CHL+05] Melissa Chase, Alexander Healy, Anna Lysyanskaya, Tal Malkin, and Leonid Reyzin. Mercurial commitments with applications to zero-knowledge sets. In Ronald Cramer, editor, *Advances in Cryptology—EUROCRYPT 2005*, volume 3494 of *Lecture Notes in Computer Science*, pages 422–439. Springer-Verlag, 2005.

[10] We remark that, obviously, a bijection between G and $\{0, 1, \ldots, q-1\}$ always exists because $|G| = q$; the reason for using Sophie Germain primes is that we do not know how construct a simple efficient bijection otherwise.

[FF02] Marc Fischlin and Roger Fischlin. The representation problem based on factoring. In *RSA Security 2002 Cryptographer's Track*, volume 2271 of *Lecture Notes in Computer Science*. Springer-Verlag, 2002.

[Fis01] Marc Fischlin. *Trapdoor Commitment Schemes and Their Applications*. PhD thesis, University of Frankfurt am Main, December 2001.

[FLS99] Uriel Feige, Dror Lapidot, and Adi Shamir. Multiple noninteractive zero knowledge proofs under general assumptions. *SIAM Journal on Computing*, 29(1):1–28, 1999.

[GM06] Rosario Gennaro and Silvio Micali. Independent zero-knowledge sets. In *33rd International Colloquium on Automata, Languages and Programming (ICALP)*, 2006.

[GMR88] Shafi Goldwasser, Silvio Micali, and Ronald Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM Journal on Computing*, 17(2):281–308, April 1988.

[GO92] Shafi Goldwasser and Rafail Ostrovsky. Invariant signatures and non-interactive zero-knowledge proofs are equivalent. In Ernest F. Brickell, editor, *Advances in Cryptology — CRYPTO '92*, pages 228–244. Springer-Verlag, 1992. Lecture Notes in Computer Science No. 740.

[HILL99] Johan Håstad, Russell Impagliazzo, Leonid A. Levin, and Michael Luby. A pseudorandom generator from any one-way function. *SIAM Journal of Computing*, 28(4):1364–1396, 1999.

[Lis05] Moses Liskov. Updatable zero-knowledge databases. In *Advances in Cryptology — ASIACRYPT 2005*, volume 3788 of *Lecture Notes in Computer Science*, pages 174–198. Springer, 2005.

[LL94] Chae Hoon Lim and Pil Joong Lee. More flexible exponentiation with precomputation. In Yvo G. Desmedt, editor, *Advances in Cryptology—CRYPTO '94*, volume 839 of *Lecture Notes in Computer Science*, pages 95–107. Springer-Verlag, 21–25 August 1994.

[Lys02a] Anna Lysyanskaya. *Signature Schemes and applications to cryptographic protocol design*. PhD thesis, Massachusetts Institute of Technology, Cambridge, Massachusetts, September 2002.

[Lys02b] Anna Lysyanskaya. Unique signatures and verifiable random functions from the DH-DDH separation. In Moti Yung, editor, *Advances in Cryptology — CRYPTO 2002*, Lecture Notes in Computer Science, pages 597–612. Springer Verlag, 2002.

[Mic] Silvio Micali. 6.875: Introduction to cryptography. MIT course taught in Fall 1997.

[MRK03] Silvio Micali, Michael Rabin, and Joe Kilian. Zero-knowledge sets. In *Proc. 44th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 80–91. IEEE Computer Society Press, 2003.

[MRV99] Silvio Micali, Michael Rabin, and Salil Vadhan. Verifiable random functions. In *Proc. 40th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 120–130. IEEE Computer Society Press, 1999.

[Nao91] Moni Naor. Bit commitment using pseudorandomness. *Journal of Cryptology*, 4(2):51–158, 1991.

[ORS04] Rafi Ostrovsky, Charles Rackoff, and Adam Smith. Efficient consistency proof on a committed database. In *Automata, Languages and Programming: 31st International Colloquium, ICALP 2004, Turku, Finland, July 12-16, 2004. Proceedings*, volume 3142 of *Lecture Notes in Computer Science*, pages 1041–1053. Springer, 2004.

[Ped92] Torben Pryds Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In Joan Feigenbaum, editor, *Advances in Cryptology – CRYPTO '91*, volume 576 of *Lecture Notes in Computer Science*, pages 129–140. Springer Verlag, 1992.

[PX07] Manoj Prabhakaran and Rui Xue. Statistically hiding sets. Cryptology ePrint Archive, Report 2007/349, 2007. `http://eprint.iacr.org/`.

A Definitions of NIZK and Conventional Commitment Schemes

Here we describe in more detail the two building blocks used in Section 2.3.1 to construct noninteractive mercurial commitments from general assumptions (the existence of the second building block is implied by the existence of the first).

NIZK The first building block is a many-theorem noninteractive zero-knowledge (NIZK) proof system for an NP-complete language L. This proof system operates in the shared random string model, and consists of polynomial-time algorithms Prove and Verify.

Here, we omit the definition of NIZK, but refer the reader to Blum et al. [BDMP91] and Feige, Lapidot, Shamir [FLS99]. Instead, we informally sketch this definition:

Algorithms Prove **and** Verify The algorithm Prove takes as input the shared random string σ, and values (x, w), such that $x \in L$, and w is a witness to this. Prove outputs a *proof* π. Verify is the algorithm that takes (σ, x, π) as input, and outputs *ACCEPT* or *REJECT*.

Completeness For all $x \in L$, for all witnesses w for x, for all values of the shared random string σ, and for all outputs π of Prove(σ, x, w), Verify$(\sigma, x, \pi) = ACCEPT$.

Soundness For all adversarial prover algorithms $\mathcal{A}$, for a randomly chosen σ, the probability that $\mathcal{A}$ can produce (x, π) such that $x \notin L$ but Verify$(\sigma, x, \pi) = ACCEPT$, is negligible.

Zero-knowledge There exist algorithms ZKSim-Setup and ZKSim-Prove, as follows: ZKSim-Setup(1^k) outputs (σ, s). For all x, ZKSim-Prove(σ, s, x) outputs a *simulated proof* π^S. Even for a sequence of adversarially and adaptively picked $(x_1, \ldots, x_m)$ (m is polynomial in k), if for all $1 \leq i \leq m$, $x_i \in L$, then the simulated proofs $\pi_1^S, \ldots, \pi_m^S$ are distributed indistinguishably from proofs $\pi_1, \ldots, \pi_m$ that are computed by running Prove(σ, x_i, w_i), where w_i's is some witness that $x_i \in L$.

Remark. Note that above we talk about a proof system (Prove, Verify) for a specific NP-complete language L. Since L is NP-complete, it follows that any language $L' \in$NP also has such a proof system. We will make use of this fact and, for example, say that the Prove and Verify algorithms work for some other language $L' \in$NP. By that we will mean that appropriate Prove and Verify algorithms for L' can be obtained from those for L.

Commitment Schemes The second building block is a conventional noninteractive unconditionally binding commitment scheme, consisting of algorithms (Comm-Setup, Commit). Note that such a commitment scheme is already implied by the existence of NIZK, because NIZK implies OWFs, and in the shared random string model, OWFs imply setup-free unconditionally binding bit commitment [Nao91, HILL99].

More formally, a noninteractive unconditionally binding commitment scheme is defined as follows:

Definition A.1. A pair of algorithms (Comm-Setup, Commit) constitute a noninteractive unconditionally binding commitment scheme if:

Algorithms Comm-Setup **and** Commit Comm-Setup is a probabilistic poly-time algorithm that takes as input the security parameter 1^k, and outputs a public key p for the commitment scheme. We use notation $p \leftarrow \textsc{Comm-Setup}(1^k)$. p implicitly defines the domain for the committed values, D_p ($|D_p| \geq 2$). Commit is a deterministic algorithm that takes as input the public key p, a value $x \in D_p$ and a random string r of length $\ell_r(k)$ and computes the commitment $C = \textsc{Commit}(p, x, r)$.

Binding property The probability, over the choice of p, that there exist (x_1, r_1) and (x_2, r_2), $x_1 \neq x_2$ such that $\textsc{Commit}(p, x_1, r_1) = \textsc{Commit}(p, x_2, r_2)$, is negligible.

Hiding property For all probabilistic polynomial-time algorithms $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$, there exists a negligible function $\nu(k)$ such that

$$\Pr[p \leftarrow \textsc{Comm-Setup}(1^k); (\mathtt{state}_A, x_0, x_1) \leftarrow \mathcal{A}_1(p); b \leftarrow \{0,1\};$$
$$r \leftarrow \{0,1\}^{\ell_r(k)}; C = \textsc{Commit}(p, x_b, r); b' \leftarrow \mathcal{A}_2(\mathtt{state}_A, C) \; : \; b = b'] \quad \leq \quad 1/2 + \nu(k)$$