Mercurial Commitments with Applications to Zero-Knowledge Sets

Extended Abstract

Melissa Chase[1], Alexander Healy[2], Anna Lysyanskaya[1], Tal Malkin[3], and Leonid Reyzin[4]

[1] Brown University
{mchase,anna}@cs.brown.edu
[2] Harvard University
ahealy@fas.harvard.edu
[3] Columbia University
tal@cs.columbia.edu
[4] Boston University
reyzin@cs.bu.edu

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. The two usual properties of commitments are therefore *hiding* and *binding*: 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 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[5], 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.

[5] 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).

We formally define mercurial commitments in Section 2.1. We provide four constructions in Section 2.2: 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.

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 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 are interesting independently of this specific application because of their potentially broader applicability.

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.[6])

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 instantiations 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 anyway implied by NIZK), and CRHF.[7] 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). 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

[6] 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 $d = k$.

[7] It is known how to construct NIZK proofs based on the existence of trapdoor permutations (TDP) [FLS99,BY96], or based on the existence of verifiable random functions (VRF) [GO92,MRV99]. TDP and VRF are, as far as we currently know, incomparable assumptions. Indeed, VRFs can be constructed based on gap-Diffie-Hellman groups [Lys02], while no trapdoor permutation is known based on such groups.

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 3.2).

2 The New Primitive: Mercurial Commitments

2.1 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 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 $\tau \leftarrow$ $\textsc{Tease}(PK, x, r, C)$.

$\textsc{Ver-Tease}$ 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.

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

$\textsc{Ver-Open}$ 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: $\textsc{Ver-Tease}$ will always accept the correctly formed partial decommitment τ of C for the correct teaser value x, and $\textsc{Ver-Open}$ 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 not only partially decommit (tease) it to any teaser value, but also open it to any value, such that the view of the receiver will be the same whether it is talking to the committer or to the simulator.

More precisely:

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

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

Definition 2. *A set of algorithms* $\textsc{Setup}$*,* $\textsc{Ver-Tease}$ *and* $\textsc{Ver-Open}$ *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*

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

Definition 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* $(PK, TK) \leftarrow \textsc{Sim-Setup}(1^k)$.

   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* $C = \textsc{Sim-Commit}(PK, TK, r)$.

   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* $C = \textsc{Sim-Commit}$ (PK, TK, r)*. We write* $\tau \leftarrow \textsc{Sim-Tease}(PK, TK, r, x)$.

   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* $C = \textsc{Sim-Commit}(PK, TK, r)$*. We write* $\pi \leftarrow \textsc{Sim-Open}(PK, TK, r, x)$.

2. *Let the following algorithms be defined as follows:*

   Committer$_{PK}$ *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* $(\textsc{Hard-Comm}, C, x, r)$ *in the list L. Output C.*
   - *On input* $(\textsc{Soft-Comm})$*, choose a random string r. Compute* $C = \textsc{Soft-Comm}(PK, r)$*. Store* $(\textsc{Soft-Comm}, C, r)$ *in the list L. Output C.*
   - *On input* $(\textsc{Tease}, C, x')$*:*
     - *Check if $C \in L$. 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$_{(PK,TK)}$ *The simulator $S_{(PK,TK)}$ answers the same types of queries as the Committer C_{PK}, but by running different algorithms. It also maintains the* same *list L, initialized to empty.*

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

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. Also note that the definition above is noninteractive. The definition can be extended to an interactive setting, where the decommitment (opening or teasing) is interactive. Throughout the paper, in order to keep the presentation clean, we continue by default to consider noninteractive mercurial commitments (and noninteractive ZKS), and only mention the interactive case in side remarks.

Definition 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.*

2.2 Constructions

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 [Lys02], 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 public 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 OWFs, and in the public-random-string model, OWFs imply setup-free unconditionally binding bit commitment [Nao91,HILL99]. More detailed definitions of these standard building blocks, NIZK and commitment schemes, are given in the full version of the paper.

We now 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.

An (easily produced) more formal description of the above scheme is contained in the full version of the paper. We thus obtain the following theorem.

Theorem 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.*

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 OWF, the construction yields interactive mercurial commitments based on OWF.

From Claw-Free Trapdoor Bijections. We now construct a mercurial bit-commitment scheme under the assumption that there exist *claw-free trapdoor bijections*.[8] 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-commitmens:

- $PK = \textsc{Setup}(1^n) = i$ where (i, t_i) is sampled from Δ
- Using randomness $(r_0, r_1) \in D_{f_i} \times D_{g_i}$, $\textsc{Hard-Comm}(i, 0, (r_0, r_1)) = (f_i(r_0), g_i(r_1))$ and $\textsc{Hard-Comm}(i, 1, (r_0, r_1)) = (g_i(r_1), f_i(r_0))$
- Using randomness $(r_0, r_1) \in D_{f_i} \times D_{f_i}$, $\textsc{Soft-Comm}(i, (r_0, r_1)) = (f_i(r_0), f_i(r_1))$
- For hard commitment $C = (C_0, C_1)$, $\tau = \textsc{Tease}(i, x, (r_0, r_1), (C_0, C_1)) = r_0$.
- For soft commitment $C = (C_0, C_1)$, $\tau = \textsc{Tease}(i, x, (r_0, r_1), (C_0, C_1)) = r_x$.
- $\textsc{Ver-Tease}(i, x, \tau, (C_0, C_1))$ 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), $\pi = \textsc{Open}(i, x, (r_0, r_1), (C_0, C_1)) = (x, r_0, r_1)$.
- Given a decommitment $\pi = (x, r_0, r_1)$, $\textsc{Ver-Open}(i, x, \pi, (C_0, C_1))$ checks $C_x = f_i(r_0)$ and $C_{1-x} = g_i(r_1)$.

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 $r \leftarrow D_{g_i}$. The details are provided in the full version of the paper.

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, the assumption seems reasonable generically: the

[8] 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 $x \in \{0,1\}$.

less the functions f_i and g_i have to do with each other, the more plausible the 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 it is infeasible to find a claw in such a case.

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].

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 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 G and $(g,h) \leftarrow G \times G$.
- The hard commitment (with random string $(r_0, r_1) \leftarrow \{0, 1, \ldots, |G|-1\}^2$) is simply the Pedersen commitment 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 $(r_0, r_1) \leftarrow \{0, 1, \ldots, |G|-1\}^2$) is $\textsc{Soft-Comm}((g,h), (r_0, r_1)) = (g^{r_0}, g^{r_1})$.
- If $C = (C_0, C_1)$ is hard, then $\tau = \textsc{Tease}((g,h), x, (r_0, r_1), (C_0, C_1)) = r_0$.
- If $C = (C_0, C_1)$ is soft, then $\tau = \textsc{Tease}((g,h), x, (r_1, r_2), (C_0, C_1)) = (r_0 - x)/r_1 \bmod |G|$.
- 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, the simulator simply needs to set up g, h and TK as $g \leftarrow G$, $TK \leftarrow \{0, 1, \ldots, |G|-1\}$ and $h = g^{TK}$. A more detailed description is contained in the full version of the paper.

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 based on the Strong RSA assumption) was independently discovered by Gennaro and Micali [GM05].

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 $p \equiv q \equiv 3 \pmod 4$, and a random element $y \in \mathbb{Z}_N^*$. Let $U = y^2$. $PK = (N, U)$.
- Using randomness $(r_0, r_1) \in \mathbb{Z}_N^* \times \mathbb{Z}_N^*$, $\textsc{Hard-Comm}((N,U), m, (r_0, r_1)) = (Ur_0^2, r_1^{2^\ell}(Ur_0^2)^m)$.
- Using randomness $(r_0, r_1) \in \mathbb{Z}_N^* \times \mathbb{Z}_N^*$, $\textsc{Soft-Comm}((N,U), (r_0, r_1)) = (r_0^{2^\ell}, r_1^{2^\ell})$.
- If $C = (C_0, C_1)$ is hard, then $\tau = \textsc{Tease}((N,U), m, (r_0, r_1), (C_0, C_1)) = r_1$.
- If $C = (C_0, C_1)$ is soft, then $\tau = \textsc{Tease}((N,U), m, (r_0, r_1), (C_0, C_1)) = r_1 r_0^{-m}$.
- $\textsc{Ver-Tease}((N,U), m, \tau, (C_0, C_1))$ checks that $C_1 = C_0^m (\tau)^{2^\ell}$.
- $\textsc{Open}$ computes π as $\pi = \textsc{Open}((N,U), m, (r_0, r_1), (C_0, C_1)) = (m, r_0, r_1)$.
- Given a decommitment $\pi = (m, r_0, r_1)$, $\textsc{Ver-Open}((N, U, \ell), m, \pi, (C_0, C_1))$ checks $C_0 = Ur_0^2$ and $C_1 = C_0^m r_1^{2^\ell}$.

The correctness of this commitment scheme follows directly from the above definitions. Simulatability follows if we simply let the simulator set up U as $U = y^{2^\ell}$. The details of the simulator are in the full version of the paper.

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 $C_0 = Ur_0^2 = Ur_0'^2$, and $C_1 = C_0^m r_1^{2^\ell} = C_0^{m'} r_1'^{2^\ell}$. Given that $m \neq m'$, this means that $r_1 \neq r_1'$. Let $f_0(x) = x^2$, $f_1(x) = C_0 x^2$. 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: $(U = (x_0 x_1^{-1} r_0)^2)$. This would then allow us to factor N. Thus, this is a claw-free pair. Note also that $C_0^m r_1^{2^\ell} = f_{b_\ell}(f_{b_{\ell-1}}(\dots(f_{b_1}(r_1))))$. Since there are two verifiable openings, this must be equal to $C_0^{m'} r_1'^{2^\ell} = f_{b_\ell'}(f_{b_{\ell-1}'}(\dots(f_{b_1'}(r_1'))))$. Let i be the smallest index such that $f_{b_i}(f_{b_{i-1}}(\dots(f_{b_1}(r_1)))) = f_{b_i'}(f_{b_{i-1}'}(\dots(f_{b_1'}(r_1'))))$. Such an i must clearly exist, and as long as $r_1' \neq r_1$ 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.

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

Note that a similar scheme using an arbitrary RSA modulus N can be created using a modified version of the commitment described in [Fis01].

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.

The formalization of this definition, which is a slight revision of the original definition of [MRK03] (in particular, it extends the original definition to allow computational zero-knowledge), is given in the full version of the paper.

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.

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 Hard-Comm$(PK, \cdot, \cdot)$ and/or 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 for lack of space and 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, simply run the prover algorithm on the database D to produce a commitment C, fixing the prover-randomness to an all-0 string. 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. 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.

Building ZKS. Below, we construct ZKS for a database D with keys of length[9] 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.[10]

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

[9] 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.

[10] 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.

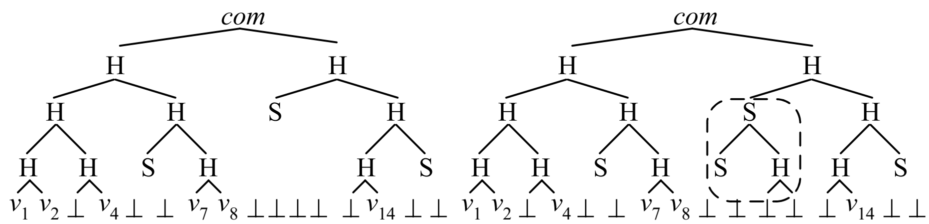

Fig. 1. A commitment tree before and after a query for key 11, whose value is not the database. The parts built in response to the query are shown in the second tree. Hard commitments are denoted by 'H' and soft commitments by 'S'. Each leaf contains a commitment to the value shown rather than the value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

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), we generate all random strings used in the commitments pseudorandomly rather than randomly.

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 full version of the paper.

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 the 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.[11] 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).

[11] 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.

Acknowledgments

The authors wish to thank Rosario Gennaro and Adi Shamir for helpful technical discussions, and Monica Perez for giving our new commitments their name "mercurial."

The authors are 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 Grant CCR-0311485.

References

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

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

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

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

[FLS99] U. Feige, D. Lapidot, and A. Shamir. Multiple noninteractive zero knowledge proofs under general assumptions. *SIAM J. Computing*, 29(1), 1999.

[GM05] Rosario Gennaro and Silvio Micali. Independent zero-knowledge sets. Unpublished manuscript, 2005.

[GMR88] S. Goldwasser, S. Micali, and R. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM J. Computing*, 17(2), 1988.

[GO92] Shafi Goldwasser and Rafail Ostrovsky. Invariant signatures and non-interactive zero-knowledge proofs are equivalent. In *CRYPTO '92*.

[HILL99] J. Håstad, R. Impagliazzo, L. Levin, and M. Luby. A pseudorandom generator from any one-way function. *SIAM J. Computing*, 28(4), 1999.

[LL94] Chae Hoon Lim and Pil Joong Lee. More flexible exponentiation with precomputation. In *Advances in Cryptology—CRYPTO '94*.

[Lys02] Anna Lysyanskaya. Unique signatures and verifiable random functions from the DH-DDH separation. In *Advances in Cryptology — CRYPTO 2002*.

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

[MRV99] S. Micali, M. Rabin, and S. Vadhan. Verifiable random functions. In *Proc. 40th IEEE Symposium on Foundations of Computer Science (FOCS)*, 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 *Proc. of Automata, Languages and Programming: 31st International Colloquium, ICALP 2004*.

[Ped92] Torben Pryds Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *Advances in Cryptology – CRYPTO '91*.