Proofs of Storage from Homomorphic Identification Protocols

Giuseppe Ateniese[1], Seny Kamara[2,⋆], and Jonathan Katz[3,⋆⋆]

[1] The Johns Hopkins University
ateniese@cs.jhu.edu
[2] Microsoft Research
senyk@microsoft.com
[3] University of Maryland
jkatz@cs.umd.edu

Abstract. Proofs of storage (PoS) are interactive protocols allowing a client to verify that a server faithfully stores a file. Previous work has shown that proofs of storage can be constructed from any homomorphic linear authenticator (HLA). The latter, roughly speaking, are signature/message authentication schemes where 'tags' on multiple messages can be homomorphically combined to yield a 'tag' on any linear combination of these messages.

We provide a framework for building public-key HLAs from any identification protocol satisfying certain homomorphic properties. We then show how to turn any public-key HLA into a publicly-verifiable PoS with communication complexity independent of the file length and supporting an unbounded number of verifications. We illustrate the use of our transformations by applying them to a variant of an identification protocol by Shoup, thus obtaining the first unbounded-use PoS based on factoring (in the random oracle model).

1 Introduction

Advances in networking technology and the rapid accumulation of information have fueled a trend toward outsourcing data management to external service providers ("servers"). By doing so, organizations can concentrate on their core tasks rather than incurring the substantial hardware, software and personnel costs involved in maintaining data "in house".

Outsourcing storage prompts a number of interesting challenges. One problem is to verify that the server continually and faithfully stores the entire file $\boldsymbol{f}$ entrusted to it by the client. The server is untrusted in terms of both security and reliability: it might maliciously or accidentally erase the data or place it onto temporarily unavailable storage media. This could occur for numerous reasons including cost-savings or external pressures (e.g., government censure).

⋆ Portions of this work done while at Johns Hopkins.

⋆⋆ Portions of this work done while at IBM. Research supported by NSF grant #0426683.

M. Matsui (Ed.): ASIACRYPT 2009, LNCS 5912, pp. 319–333, 2009.

The server might also accidentally erase some data and choose not to notify the client. Exacerbating the problem (and precluding naïve approaches) are factors such as limited bandwidth between the client and server, as well as the client's limited resources. See [1,11] for a more thorough discussion.

If we allow communication complexity linear in $\boldsymbol{f}$, there is a simple mechanism allowing the client to verify that the server stores $\boldsymbol{f}$ at any given time: When the client uploads $\boldsymbol{f}$, the client locally stores a hash of $\boldsymbol{f}$; to verify, the server simply sends all of $\boldsymbol{f}$ and the client checks that this hashes to the correct value. For our purposes, we are interested in solutions with communication complexity that is much smaller than (and, ideally, independent of) the file size.

Ateniese et al. [1] and Juels and Kaliski [11] independently introduced approaches to this problem having sub-linear communication complexity. (Earlier work by Naor and Rothblum [13] is related, but considers a somewhat weaker adversarial model.) Ateniese et al. also distinguish between the case of *private verifiability*, where only the original client (or anyone with whom that client shares a key) can verify the server's storage, and *public verifiability*, where anyone knowing the client's public key can perform verification. Extensions and improvements were given by Shacham and Waters [14], Dodis, Vadhan, and Wichs [5], and Bowers, Juels, and Oprea [4]. We refer to [5] for a more detailed comparison among the existing schemes.

Here, we are interested in *publicly-verifiable* schemes that can be used for an *unbounded* number of verifications. A useful tool for this, implicit in [1] and further studied in [14,5], is a *homomorphic linear authenticator* (HLA), which can be defined in either the private- or public-key setting. Roughly speaking, this primitive allows a client to 'tag' each block f_i of a file $\boldsymbol{f} = f_1|\cdots|f_n$ in such a way that for any vector $\boldsymbol{c}$ the server can homomorphically construct a (short) tag authenticating the value $\sum c_i \cdot f_i$.

Two recent works have considered the dynamic setting, where the remotely-stored data can be updated [2,6]. We do not address this problem here.

1.1 Our Contributions

The main contribution of this paper is to show a general mechanism (in the random oracle model) for constructing publicly-key HLAs from any identification protocol that is suitably homomorphic. The RSA-based HLA used by Ateniese et al. [1] (see also [14, Appendix E]) can be viewed as an instance of our mechanism applied to the Guillou-Quisquater [10] identification protocol; similarly, the Shacham-Waters scheme [14] can be seen as being derived from an underlying identification protocol in bilinear groups. By applying our transformation to a variant of Shoup's identification scheme based on factoring [15], we obtain the first publicly-verifiable HLA based on factoring (in the random oracle model).

We also show a generic transformation from any HLA to a publicly-verifiable proof of storage with communication complexity independent of the file size. This transformation is in the standard model, and answers an open question from [14]. An analogous transformation with similar properties was shown (independently)

by Dodis et al. [5] in the setting of simpler private verifiability; our technique is different from theirs and is of independent interest.

Combining our results, we obtain a publicly-verifiable proof of storage based on the factoring assumption in the random oracle model. In our PoS, the communication complexity and the size of the client's state are independent[1] of the file size, and the server's storage is a constant multiple of the file size. In the PoS we describe, the computation of both the client and the server is linear in the file size, but notice that public-key HLAs can be layered on top of erasure codes (as in [14,4]) or used in conjunction with a probabilistic approach for multiple audits (as in [1]) to obtain better performance while retaining public verifiability.

2 Definitions

We write $x \leftarrow X$ to represent an element x being sampled uniformly at random from a set X. The output y of a randomized algorithm $\mathcal{A}$ running on input x is denoted by $x \leftarrow \mathcal{A}(x)$. We sometimes write $y := \mathcal{A}(x; r)$ to denote the (deterministic) result of running $\mathcal{A}$ on input x and random coins r. We use boldface to denote vectors. Given a vector $\boldsymbol{v}$ we let v_i denote its ith component.

Throughout, $k \in \mathbb{N}$ denotes the security parameter. A function $\nu : \mathbb{N} \to \mathbb{R}$ is *negligible* if for every polynomial $p(\cdot)$ and large enough k, we have $\nu(k) < 1/p(k)$.

2.1 Homomorphic Linear Authenticators

Homomorphic linear authenticators (HLAs) were introduced by Ateniese et al. [1] as a building block for constructing communication-efficient proofs of storage; they were further studied in [14,5]. At a high level, HLAs are used as follows: viewing the file $\boldsymbol{f}$ as an n-dimensional vector, the client begins by tagging each element of $\boldsymbol{f}$ and then sending both $\boldsymbol{f}$ and the vector of tags $\boldsymbol{t}$ to the server. To verify that the server is storing the entire file, the client sends a random challenge vector $\boldsymbol{c}$ and the server returns $\mu = \sum_i c_i \cdot f_i$ along with a tag τ, computed using $\boldsymbol{f}, \boldsymbol{t}$, and $\boldsymbol{c}$, which is supposed to authenticate this value.

HLAs can be defined both in the private and public-key settings. We give a definition for public-key HLAs and refer the reader to [5] for a formalization of private-key HLAs.

Definition 1 (Homomorphic linear authenticator). *A* public-key homomorphic linear authenticator *is a tuple of four* PPT *algorithms* $(\mathsf{Gen}, \mathsf{Tag}, \mathsf{Auth}, \mathsf{Vrfy})$ *such that:*

- $(pk, sk) \leftarrow \mathsf{Gen}(1^k)$ *is a probabilistic algorithm used to set up the scheme. It takes as input the security parameter and outputs a public and private key pair* (pk, sk)*. We assume* pk *defines a* k*-bit prime* p *and a positive integer* B*.*
- $(\boldsymbol{t}, st) \leftarrow \mathsf{Tag}_{sk}(\boldsymbol{f})$ *is a probabilistic algorithm that is run by the client in order to tag a file. It takes as input a secret key* sk *and a file* $\boldsymbol{f} \in [B]^n$*, and outputs a vector of tags* $\boldsymbol{t}$ *and state information* st*.*

[1] The communication complexity for a file of size n is $\mathcal{O}(\log n + k)$, and as in [5] we assume $k \gg \log n$.

$\tau := \mathsf{Auth}_{pk}(\boldsymbol{f}, \boldsymbol{t}, \boldsymbol{c})$ *is a deterministic algorithm that is run by the server to generate a tag. It takes as input a public key* pk*, a file* $\boldsymbol{f} \in [B]^n$*, a tag vector* $\boldsymbol{t}$*, and a challenge vector* $\boldsymbol{c} \in \mathbb{Z}_p^n$*; it outputs a tag* τ*.*

$b := \mathsf{Vrfy}_{pk}(st, \mu, \boldsymbol{c}, \tau)$*: is a deterministic algorithm that is used to verify a tag. It takes as input a public key* pk*, state information* st*, an element* $\mu \in \mathbb{N}$*, a challenge vector* $\boldsymbol{c} \in \mathbb{Z}_p^n$*, and a tag* τ*. It outputs a bit, where '1' indicates acceptance and '0' indicates rejection.*

For correctness, we require that for all $k \in \mathbb{N}$*, all* (pk, sk) *output by* $\mathsf{Gen}(1^k)$*, all* $\boldsymbol{f} \in [B]^n$*, all* $(\boldsymbol{t}, st)$ *output by* $\mathsf{Tag}_{sk}(\boldsymbol{f})$*, and all* $\boldsymbol{c} \in \mathbb{Z}_p^n$*, it holds that*

$$\mathsf{Vrfy}_{pk}\left(st,\ \sum_i c_i f_i,\ \boldsymbol{c},\ \mathsf{Auth}_{pk}(\boldsymbol{f}, \boldsymbol{t}, \boldsymbol{c})\right) = 1.$$

We remark that in certain schemes correctness (and security) may hold even when Vrfy is given only $\sum_i c_i f_i \bmod p$ (assuming $B < p$). In such cases the communication from the server to the client can be further reduced.

Informally an HLA is secure if, for a given file $\boldsymbol{f}$ and challenge vector $\boldsymbol{c}$, no adversary can output a valid authenticator for an element $\mu' \neq \sum_i c_i f_i$.

Definition 2 (Unforgeability for public-key HLAs). *Let* $\Lambda = (\mathsf{Gen}, \mathsf{Tag}, \mathsf{Auth}, \mathsf{Vrfy})$ *be a public-key HLA and* $\mathcal{A}$ *be an adversary, and consider the following experiment:*

1. *The challenger computes* $(pk, sk) \leftarrow \mathsf{Gen}(1^k)$*, where* pk *defines* p *and* B*.*
2. *Given* pk *and oracle access to* $\mathsf{Tag}_{sk}(\cdot)$*, adversary* $\mathcal{A}$ *outputs a file* $\boldsymbol{f} \in [B]^n$*.*
3. *The challenger tags the file by computing* $(\boldsymbol{t}, st) \leftarrow \mathsf{Tag}_{sk}(\boldsymbol{f})$*.*
4. *Given* $\boldsymbol{t}$ *and* st*, the adversary* $\mathcal{A}$ *outputs a challenge vector* $\boldsymbol{c} \in \mathbb{Z}_p^n$*, an element* $\mu' \in \mathbb{Z}$*, and a tag* τ'*.*
5. *The adversary succeeds if* $\mu' \neq \sum_i c_i f_i$ *and* $\mathsf{Vrfy}_{pk}(st, \mu', \boldsymbol{c}, \tau') = 1$*.*

Λ *is* unforgeable *if the success probability of every* PPT *adversary* $\mathcal{A}$ *in the above experiment is negligible.*

The distinctions between the case of public verifiability (as defined above) and private verifiability (as defined in [5]) are that, in the former setting (1) verification does not require the original secret key sk but only the state st and the original public key; (2) unforgeability holds even against an adversary who knows the public information pk and st. Our definition is also stronger than the one given in [5] in that we initially give the adversary access to a tagging oracle.

2.2 Homomorphic Identification Protocols

An identification protocol allows a prover $\mathcal{P}$ in possession of a secret key sk to prove its identity to a verifier $\mathcal{V}$ that possesses the corresponding public key pk. We consider 3-move identification protocols where the prover generates the first message α using the public key pk and randomness r; the verifier sends a random challenge β; and the prover then computes a response γ using (pk, sk), the randomness r, and the verifier's challenge β. Given the transcript of the protocol, the verifier decides whether to accept or not.

Definition 3 (Identification protocol). *An* identification protocol *is a three-move protocol between a* PPT *prover* $\mathcal{P}$ *and a* PPT *verifier* $\mathcal{V}$*. The protocol consists of four polynomial-time algorithms* $(\mathsf{Setup}, \mathsf{Comm}, \mathsf{Resp}, \mathsf{Vrfy})$ *such that:*

- $(pk, sk) \leftarrow \mathsf{Setup}(1^k)$ *is a probabilistic algorithm that takes as input the security parameter and outputs a public and private key pair* (pk, sk)*.*
- $\alpha \leftarrow \mathsf{Comm}(pk; r)$ *is a probabilistic algorithm run by the prover* $\mathcal{P}$ *to generate the first message. It takes as input the public key and random coins* r*, and outputs an initial message* α*. We stress that there is no need for* sk*.*
- $\gamma \leftarrow \mathsf{Resp}(pk, sk, r, \beta)$ *is a probabilistic algorithm that is run by the prover* $\mathcal{P}$ *to generate the third message. It takes as input the public key* pk*, the secret key* sk*, a random string* r*, and a challenge* β *(from some associated challenge space), and outputs a response* γ*.*
- $b := \mathsf{Vrfy}(pk, \alpha, \beta, \gamma)$ *is a deterministic algorithm run by the verifier* $\mathcal{V}$ *to decide whether to accept the interaction. It takes as input the public key* pk*, an initial message* α*, a challenge* β*, and a response* γ*. It outputs a bit* b*, where '1' indicates acceptance and '0' indicates rejection.*

For correctness, we require that for all $k \in \mathbb{N}$*, all* (pk, sk) *output by* $\mathsf{Setup}(1^k)$*, all random coins* r*, and all* β *in the appropriate challenge space, it holds that*

$$\mathsf{Vrfy}\Big(pk, \mathsf{Comm}(pk; r), \beta, \mathsf{Resp}(pk, sk, r, \beta)\Big) = 1.$$

An identification protocol is *homomorphic* if the verification of several transcripts of the protocol can be "batched":

Definition 4 (Homomorphic identification protocol). *An identification protocol* $\Sigma = (\mathsf{Setup}, \mathsf{Comm}, \mathsf{Resp}, \mathsf{Vrfy})$ *is* homomorphic *if there exist efficient functions* $\mathsf{Combine}_1, \mathsf{Combine}_3$ *such that:*

Completeness: *For all* (pk, sk) *output by* $\mathsf{Setup}(1^k)$ *and all* $\boldsymbol{c} \in \mathbb{Z}_{2^k}^n$*, if transcripts* $\{(\alpha_i, \beta_i, \gamma_i)\}_{1 \le i \le n}$ *are such that* $\mathsf{Vrfy}(pk, \alpha_i, \beta_i, \gamma_i) = 1$ *for all* i*, then:*

$$\mathsf{Vrfy}\left(pk,\ \mathsf{Combine}_1(\boldsymbol{c}, \boldsymbol{\alpha}),\ \sum_i c_i \beta_i,\ \mathsf{Combine}_3(\boldsymbol{c}, \boldsymbol{\gamma})\right) = 1.$$

Unforgeability: *Consider the following experiment involving an adversary* $\mathcal{A}$*:*

1. *The challenger computes* $(pk, sk) \leftarrow \mathsf{Setup}(1^k)$ *and gives* pk *to* $\mathcal{A}$*.*
2. *The following is repeated a polynomial number of times:*
   - $\mathcal{A}$ *outputs* β' *in the challenge space. The challenger chooses random* r*, computes* $\gamma := \mathsf{Resp}(pk, sk, r, \beta')$*, and gives* (r, γ) *to* $\mathcal{A}$*.*
3. *The adversary outputs a* n*-vector of challenges* $\boldsymbol{\beta}$*. Then for each* i *the challenger chooses* r_i *at random, sets* $\alpha_i := \mathsf{Comm}(pk; r_i)$ *and* $\gamma_i := \mathsf{Resp}(pk, sk, r_i, \beta_i)$*, and gives* $(\boldsymbol{r}, \boldsymbol{\gamma})$ *to* $\mathcal{A}$*.*
4. $\mathcal{A}$ *outputs a triple* $(\boldsymbol{c}, \mu', \gamma')$*, where* $\boldsymbol{c} \in \mathbb{Z}_{2^k}^n$*. The adversary succeeds if (1)* $\mu' \neq \sum_i c_i \beta_i$ *and (2)* $\mathsf{Vrfy}(pk, \mathsf{Combine}_1(\boldsymbol{c}, \boldsymbol{\alpha}), \mu', \gamma') = 1$*.*

2.3 Proofs of Storage

Definition 5 (Proof of storage). *A (publicly-verifiable)* proof of storage *is a tuple of five* PPT *algorithms* $(\mathsf{Gen}, \mathsf{Encode}, \mathsf{Prove}, \mathsf{Vrfy})$ *such that:*

- $(pk, sk) \leftarrow \mathsf{Gen}(1^k)$ *is a probabilistic algorithm that is run by the client to set up the scheme. It takes as input a security parameter, and outputs a public and private key pair* (pk, sk)*. We assume* pk *defines a* k*-bit prime* p *and a positive integer* B*.*
- $(\boldsymbol{f}', st) \leftarrow \mathsf{Encode}_{sk}(\boldsymbol{f})$ *is a probabilistic algorithm that is run by the client in order to encode the file. It takes as input the secret key* sk*, and a file* $\boldsymbol{f} \in [B]^n$*. It outputs an encoded file* $\boldsymbol{f}'$ *and state information* st*.*
- $\pi := \mathsf{Prove}(pk, \boldsymbol{f}', \boldsymbol{c})$ *is a deterministic algorithm that takes as input the public key* pk*, an encoded file* $\boldsymbol{f}'$*, and a challenge* $\boldsymbol{c} \in \mathbb{Z}_p^n$*. It outputs a proof* π*.*
- $b := \mathsf{Vrfy}(pk, st, \boldsymbol{c}, \pi)$*: is a deterministic algorithm that takes as input the public key* pk*, the state* st*, a challenge* $\boldsymbol{c} \in \mathbb{Z}_p^n$*, and a proof* π*. It outputs a bit, where '1' indicates acceptance and '0' indicates rejection.*

We require that for all $k \in \mathbb{N}$*, all* (pk, sk) *output by* $\mathsf{Gen}(1^k)$*, all* $\boldsymbol{f} \in [B]^n$*, all* $(\boldsymbol{f}', st)$ *output by* $\mathsf{Encode}_{sk}(\boldsymbol{f})$*, and all* $\boldsymbol{c} \in \mathbb{Z}_p^n$*, it holds that*

$$\mathsf{Vrfy}\left(pk, st, \boldsymbol{c}, \mathsf{Prove}(pk, \boldsymbol{f}', \boldsymbol{c})\right) = 1.$$

Note that the above defines a *publicly-verifiable* PoS since the original secret key sk is not needed in order to perform verification.

Security of a PoS, roughly speaking, guarantees that if the verifier accepts then the prover indeed has (sufficient information to recover) the entire original file $\boldsymbol{f}$. As noted in [1,11,14,5], soundness can be formalized using the notion of a knowledge extractor [7,3]. As in [5], we phrase our definition using the paradigm of "witness-extended emulation" [12].

Definition 6 (Security for a publicly-verifiable PoS). *Let* $\Pi = (\mathsf{Gen}, \mathsf{Encode}, \mathsf{Prove}, \mathsf{Vrfy})$ *be a publicly-verifiable PoS.* Π *is* secure *if there is an expected polynomial-time knowledge extractor* $\mathcal{K}$ *such that, for any* PPT *adversary* $\mathcal{A}$ *we have:*

1. *The distributions*

$$\left\{ \begin{array}{c} (pk, sk) \leftarrow \mathsf{Gen}(1^k); (\boldsymbol{f}, st_{\mathcal{A}}) \leftarrow \mathcal{A}^{\mathsf{Encode}_{sk}(\cdot)}(pk); \\ (\boldsymbol{f}', st) \leftarrow \mathsf{Encode}_{sk}(\boldsymbol{f}); \boldsymbol{c} \leftarrow \mathbb{Z}_p^n \end{array} : (\boldsymbol{c}, \mathcal{A}(st_{\mathcal{A}}, \boldsymbol{f}', st, \boldsymbol{c})) \right\}$$

   and

$$\left\{ \begin{array}{c} (pk, sk) \leftarrow \mathsf{Gen}(1^k); (\boldsymbol{f}, st_{\mathcal{A}}) \leftarrow \mathcal{A}^{\mathsf{Encode}_{sk}(\cdot)}(pk); \\ (\boldsymbol{f}', st) \leftarrow \mathsf{Encode}_{sk}(\boldsymbol{f}) \end{array} : \mathcal{K}_1^{\mathcal{A}(st_{\mathcal{A}}, \boldsymbol{f}', st, \cdot)}(pk, st) \right\}$$

   are identical. (Above, $\mathcal{K}_1$ *denotes the first output of* $\mathcal{K}$*.)*
2. *The following is negligible:*

$$\Pr\left[\begin{array}{c} (pk, sk) \leftarrow \mathsf{Gen}(1^k); \\ (\boldsymbol{f}, st_{\mathcal{A}}) \leftarrow \mathcal{A}^{\mathsf{Encode}_{sk}(\cdot)}(pk); \\ (\boldsymbol{f}', st) \leftarrow \mathsf{Encode}_{sk}(\boldsymbol{f}); \\ ((\boldsymbol{c}, \pi), \boldsymbol{f}^*) \leftarrow \mathcal{K}^{\mathcal{A}(st_{\mathcal{A}}, \boldsymbol{f}', st, \cdot)}(pk, st) \end{array} : \mathsf{Vrfy}(pk, st, \boldsymbol{c}, \pi) = 1 \bigwedge \boldsymbol{f}^* \neq \boldsymbol{f} \right].$$

3 From Homomorphic Identification Protocols to HLAs

We now show how to transform any homomorphic identification protocol $\Sigma = (\mathsf{Setup}, \mathsf{Comm}, \mathsf{Resp}, \mathsf{Vrfy})$ into a public-key HLA. The basic idea is to use the file blocks $f_1, \ldots, f_n$ as the "challenges" in n parallel invocations of the identification protocol. Thus, a very basic PoS would be as follows:

- The client computes $(pk, sk) \leftarrow \mathsf{Gen}(1^k)$.
- For each block f_i of the file, the client computes α_i, γ_i such that $(\alpha_i, f_i, \gamma_i)$ is an accepting transcript in the underlying identification scheme.
- The client sends to the server the file $\boldsymbol{f} = f_1 | \cdots | f_n$ and the tags $\gamma_1, \ldots, \gamma_n$; the client stores $\alpha_1, \ldots, \alpha_n$ as its own local state.

To verify that the server stores the ith block of the file, the client requests the server to send (f_i, γ_i); the client can authenticate this response by checking that $(\alpha_i, f_i, \gamma_i)$ is an accepting transcript.

There are several drawbacks to the above approach. First, the client's state is linear in the file size.[2] This is easy to remedy by having the client generate each α_i using a pseudorandom function (if private verifiability suffices) or a random oracle (if public verifiability is desired, as here). A more serious problem is that a server can easily "cheat" without being caught "too often" by throwing away blocks of the file. If the server deletes, say, 1 block from the file then it is only caught with probability $1/n$. This can be addressed, to some extent, by having the client request many blocks but then the communication complexity increases.

Instead, we rely on the *homomorphic* property of the identification scheme to "batch" the authentication of multiple blocks. Specifically, the client will send a random integer vector $\boldsymbol{c}$ and the server will respond with $\mu' := \sum_i c_i f_i$ and $\gamma' := \mathsf{Combine}_3(\boldsymbol{c}, \boldsymbol{\gamma})$; This response can be verified by checking whether

$$\mathsf{Vrfy}(pk, \mathsf{Combine}_1(\boldsymbol{c}, \boldsymbol{\alpha}), \mu', \gamma') \stackrel{?}{=} 1.$$

(See Figure 1.) Although the client-to-server communication is large, the server-to-client communication is essentially independent of the file size (cf. footnote 1). We reduce the client-to-server communication when we construct a PoS in the next section.

Theorem 1. *If Σ is an unforgeable homomorphic identification protocol, then Λ as in Figure 1 is an unforgeable public-key HLA if H is modeled as a random oracle.*

Proof. Correctness is easy to verify, and so we consider security. Let $\mathcal{A}$ be a PPT adversary attacking Λ. We construct an adversary $\mathcal{A}'$ attacking Σ as follows:

1. $\mathcal{A}'$ is given a public key pk, generates B and p in the obvious way, and runs $\mathcal{A}(pk, p, B)$.

[2] In some cases linear state may be acceptable, as long as the state is a constant fraction *shorter* than the file itself. When using certain homomorphic identification schemes, including the one discussed in Section 5, this indeed can be achieved.

Let $\Sigma = (\mathsf{Setup}, \mathsf{Comm}, \mathsf{Resp}, \mathsf{Vrfy})$ be a homomorphic identification protocol and let H be a function. Construct a public-key HLA $\Lambda = (\mathsf{Gen}, \mathsf{Tag}, \mathsf{Auth}, \mathsf{Vrfy})$ as follows:

- $\mathsf{Gen}(1^k)$: Compute $(pk, sk) \leftarrow \Sigma.\mathsf{Setup}(1^k)$. Let B be such that $[B]$ is in the challenge space of Σ, and choose a k-bit prime p. Output the public key (pk, p, B) and secret key sk.
- $\mathsf{Tag}_{sk}(\boldsymbol{f})$, where $\boldsymbol{f} = f_1|\cdots|f_n$, and $f_i \in [B]$ for all i:
  1. Choose $st \leftarrow \{0,1\}^k$.
  2. For $1 \le i \le n$:
     a. Set $r_i := H(st; i)$ and $\alpha_i := \Sigma.\mathsf{Comm}(pk; r_i)$.
     b. Compute $\gamma_i := \Sigma.\mathsf{Resp}(pk, sk, r_i, f_i)$.
  3. Output $\boldsymbol{t} := (\gamma_1, \ldots, \gamma_n)$ and st.
- $\mathsf{Auth}_{pk}(\boldsymbol{f}, \boldsymbol{t}, \boldsymbol{c})$: Compute and output $\tau \leftarrow \Sigma.\mathsf{Combine}_3(\boldsymbol{c}, \boldsymbol{t})$.
- $\mathsf{Vrfy}_{pk}(st, \mu, \boldsymbol{c}, \tau)$:
  1. for $1 \le i \le n$, set $r_i := H(st; i)$ and $\alpha_i := \Sigma.\mathsf{Comm}(pk; r_i)$.
  2. Output $\Sigma.\mathsf{Vrfy}(pk, \mathsf{Combine}_1(\boldsymbol{c}, \boldsymbol{\alpha}), \mu, \tau)$.

Fig. 1. Transforming a homomorphic identification protocol into a HLA

2. When $\mathcal{A}$ requests $\mathsf{Tag}_{sk}(\boldsymbol{f})$ for $\boldsymbol{f} = f_1|\cdots|f_n$, then (for $i = 1$ to n) $\mathcal{A}'$ queries f_i to its own oracle and receives in return (r_i, γ_i). Then $\mathcal{A}'$ chooses random $st \in \{0,1\}^k$, sets answers to the random oracle appropriately, and gives $(\gamma_1, \ldots, \gamma_n)$ and st to $\mathcal{A}$.
3. Eventually, $\mathcal{A}$ outputs a file $\boldsymbol{f}$. Following this, $\mathcal{A}'$ outputs the vector of n challenges $\boldsymbol{f} = f_1|\cdots|f_n$, and receives in return $(\boldsymbol{r}, \boldsymbol{\gamma})$. Then $\mathcal{A}'$ chooses random $st \in \{0,1\}^k$, sets[3] answers to the random oracle appropriately, and gives $(\boldsymbol{\gamma}, st)$ to $\mathcal{A}$.
4. When $\mathcal{A}$ finally outputs $\boldsymbol{c}, \mu', \tau'$, then $\mathcal{A}'$ outputs these same values.

It is easy to see that $\mathcal{A}$ succeeds in attacking Λ exactly when $\mathcal{A}'$ succeeds in attacking Σ.

4 From HLAs to Efficient Proofs of Storage

In this section we show how to use any HLA to construct a PoS having communication complexity independent of the file size. Our transformation is in the standard model.

It is immediate how an HLA can be used to construct a PoS with communication complexity linear in the file size: When storing a file $\boldsymbol{f}$, the client computes tags on all the file blocks and gives to the server the vector of tags $\boldsymbol{t}$ (along with $\boldsymbol{f}$ itself). To verify, the client chooses a random $\boldsymbol{c} \in \mathbb{Z}_p^n$ and sends it to the server; the server responds with $\sum_i c_i f_i$ and $\mathsf{Auth}_{pk}(\boldsymbol{f}, \boldsymbol{t}, \boldsymbol{c})$ (which is authenticated by

[3] We assume for simplicity that no $st \in \{0,1\}^k$ is chosen twice throughout the experiment, since this occurs with only negligible probability.

the client in the obvious way). If authentication tags output by Auth have length $\mathcal{O}(k)$, then the server-to-client communication for an n-block file is bounded by

$$\mathcal{O}(k) + \log\left(\sum_i c_i f_i\right) \leq \mathcal{O}(k) + \log n \cdot p \cdot B = \mathcal{O}(k) + \log n.$$

For typical values of k, n, this means that the server-to-client communication is (essentially) independent of the file size.

To reduce the client-to-server communication, we use a pseudorandom function F: the client sends a key $K \in \{0,1\}^k$, and the server then derives the challenge vector $\boldsymbol{c}$ by setting $c_i := F_K(i)$ for all i. (See Figure 2.) This approach is, perhaps, quite "natural",[4] but it turns out to be highly non-trivial to prove that it is sound. (This difficulty was mentioned in [14,5].) The issue is that since the key K is public, we cannot reduce to the security of the pseudorandom function in the usual way. Instead we must use a more careful analysis.

Let $\Lambda = (\mathsf{Gen}, \mathsf{Tag}, \mathsf{Auth}, \mathsf{Vrfy})$ be a public-key HLA, and let F be a pseudorandom function. Construct a publicly-verifiable PoS $\Pi = (\mathsf{Gen}, \mathsf{Encode}, \mathsf{Prove}, \mathsf{Vrfy})$ as follows:

- $\mathsf{Gen}(1^k)$: Compute and output $(pk, sk) \leftarrow \Lambda.\mathsf{Gen}(1^k)$. Let p be the prime implicit in pk.
- $\mathsf{Encode}_{sk}(\boldsymbol{f})$: Compute $(\boldsymbol{t}, st) \leftarrow \Lambda.\mathsf{Tag}_{sk}(\boldsymbol{f})$, and output $\boldsymbol{f}' = (\boldsymbol{f}, \boldsymbol{t})$ and st.
- $\mathsf{Prove}(pk, \boldsymbol{f}', K)$, where $K \in \{0,1\}^k$:
  1. Parse $\boldsymbol{f}'$ as $(\boldsymbol{f}, \boldsymbol{t})$.
  2. For $1 \leq i \leq n$ let $c_i := F_K(i)$, where c_i is viewed as an element of $\mathbb{Z}_p$.
  3. Compute $\tau \leftarrow \Lambda.\mathsf{Auth}_{pk}(\boldsymbol{f}, \boldsymbol{t}, \boldsymbol{c})$ and $\mu := \sum_i c_i f_i$.
  4. Output $\pi := (\mu, \tau)$.
- $\mathsf{Vrfy}(pk, st, K, \pi)$:
  1. Parse π as (μ, τ).
  2. For $1 \leq i \leq n$, let $c_i := F_K(i)$.
  3. Output $b := \Lambda.\mathsf{Vrfy}_{pk}(st, \mu, \boldsymbol{c}, \tau)$.

Fig. 2. Transforming an HLA into a PoS

Theorem 2. *Let Λ be an unforgeable public-key HLA, and let F be a pseudorandom function secure against non-uniform polynomial-time adversaries. Then Π as in Figure 2 is a secure publicly-verifiable PoS.*

Proof. Correctness of the construction is easily verified, and so we turn to proving security. We describe a knowledge extractor $\mathcal{K}$ that runs in expected polynomial-time and satisfies Definition 6. Recall that $\mathcal{K}$ is given pk, st as input and has

[4] A similar approach, based on pseudorandom generators, was proposed in [9] in the context of verifiable shuffles.

oracle access to $\mathcal{A}(st_{\mathcal{A}}, \boldsymbol{f}', st, \cdot)$, which we abbreviate as $\mathcal{A}(\cdot)$. Define $\boldsymbol{c}(K) = (F_K(1), \ldots, F_K(n))$. The high-level structure of $\mathcal{K}$ is as follows:

1. $\mathcal{K}$ chooses random $K \leftarrow \{0,1\}^k$ and runs $\mathcal{A}(K)$ to obtain a proof π. If $\mathsf{Vrfy}(pk, st, K, \pi) = 0$ then $\mathcal{K}$ outputs $((K, \pi), \perp)$ and stops. Otherwise, its first output will still be (K, π) but it attempts to recover the original file as described next.
2. $\mathcal{K}$ repeatedly rewinds $\mathcal{A}$ and sends it different challenges until $\mathcal{A}$ responds correctly to a total of n challenges $K_1, \ldots, K_n$ such that $\boldsymbol{c}(K_1), \ldots, \boldsymbol{c}(K_n)$ are linearly independent (over $\mathbb{Q}$). Given n successful responses to these n challenges, $\mathcal{K}$ reconstructs a candidate file $\boldsymbol{f}$, and outputs it.

The above neglects some technical details that we now formalize. If $\mathcal{A}(K)$ outputs a proof $\pi = (\mu, \tau)$ for which $\mathsf{Vrfy}_{pk}(st, \mu, \boldsymbol{c}(K), \tau) = 1$, then we say that K is a *good* challenge. $\mathcal{K}$ implements step 2, above, as follows:

1. Initialize sets $\mathsf{Good}_K := \mathsf{Good}_{\boldsymbol{c}} := \emptyset$. Keep track of the total number of calls to $\mathcal{A}$, and halt execution with output fail if 2^k calls are made.
2. Estimate the probability $\tilde{p^*}$ with which a random key K is good by running $\mathcal{A}$ with a random challenge until some fixed polynomial number $q = q(k)$ successful verifications occur. By appropriate choice of q, it is possible to ensure that the estimate $\tilde{p^*}$ is within a factor of 2 of the true probability with all but negligible probability 2^{-k^2}.
3. For $j = 1$ to n do:
   - Repeatedly sample K_j uniformly, querying $\mathcal{A}$ on each one, until a good K_j with $\boldsymbol{c}(K_j) \notin \mathsf{span}(\mathsf{Good}_{\boldsymbol{c}})$ is found. If found, then add K_j to Good_K and add $\boldsymbol{c}_j = \boldsymbol{c}(K_j)$ to $\mathsf{Good}_{\boldsymbol{c}}$, and go to the next value of j. If no such K_j is found in at most $k^2/\tilde{p^*}$ tries, then output fail and halt.
4. Let $\mathsf{Good}_K = \{K_1, \ldots, K_n\}$ and $\mathsf{Good}_{\boldsymbol{c}} = \{\boldsymbol{c}_1, \ldots, \boldsymbol{c}_n\}$, where $\boldsymbol{c}_j = \boldsymbol{c}(K_j)$, and let $\pi_j = (\mu_j, \tau_j)$ be the output of $\mathcal{A}(K_j)$. Set up the system of linear equations $\{\sum_i c_{j,i} \cdot f_i = \mu_j\}_{1 \le j \le n}$ in the unknowns $\boldsymbol{f} = (f_1, \ldots, f_n)$. Solve for $\boldsymbol{f}$ (over the integers) and output it.

We refer to the above as the *extraction subroutine*.

To complete the proof, we need to show three things. First, that $\mathcal{K}$ runs in expected polynomial time for any $\mathcal{A}$. Second, that if $\mathcal{A}$ successfully convinces a verifier in the PoS protocol with sufficiently high probability, then the extraction procedure will successfully complete (specifically, step 3 will be successful) with overwhelming probability. Third, that with overwhelming probability the file $\boldsymbol{f}$ output by the extraction procedure is indeed equal to the true file $\boldsymbol{f}$. The first and third of these items are essentially standard. The second step would be relatively straightforward if the challenge in the PoS protocol were a random vector $\boldsymbol{c}$; what makes it more complicated is that the challenge is a PRF key K that is expanded to a vector $\boldsymbol{c} = \boldsymbol{c}(K)$.

Fixing $st_{\mathcal{A}}$, $\boldsymbol{f}'$, and st, we let p^* denote the probability that a random challenge K is good; i.e., this is the probability with which $\mathcal{A}(st_{\mathcal{A}}, \boldsymbol{f}', st, \cdot)$ responds correctly to the verifier's challenge (we assume $st_{\mathcal{A}}$ includes $\mathcal{A}$'s coins).

Claim. $\mathcal{K}$ runs in expected polynomial time.

Proof. If $p^* = 0$ then it is clear that $\mathcal{K}$ runs in expected polynomial time. So assume $p^* > 0$. We must then analyze the expected running time of the extraction procedure, following [8,12]. Steps 1 and 4 take strict polynomial time. The expected running time of step 2 is exactly (some polynomial times) $q(k)/p^*$. As for step 3, there are two cases: If $\tilde{p^*} \leq p^*/2$, then the only thing we can claim is that the running time is bounded by (some polynomial times) 2^k, due to the counter being maintained in step 1. But the probability that $\tilde{p^*} \leq p^*/2$ is at most 2^{-k^2}. On the other hand, if $\tilde{p^*} > p^*/2$ then the expected running time of step 4 is at most (some polynomial times) $n \cdot k^2/\tilde{p^*} < 2nk^2/p^*$.

$\mathcal{K}$ only runs the extraction procedure with probability p^*. Thus, the overall expected running time of $\mathcal{K}$ is upper-bounded by

$$p^* \cdot \left(\mathsf{poly}(k) + \mathsf{poly}(k) \cdot q(k)/p^* + \mathsf{poly}(k) \cdot 2^k \cdot 2^{-k^2} + \mathsf{poly}(k) \cdot 2nk^2/p^* \right),$$

which is polynomial.

Claim. There exists a negligible function $\epsilon(\cdot)$ such that if $p^* > \epsilon(k)$ then the probability (conditioned on the extraction procedure being run) that the extraction procedure outputs fail is negligible.

Observe this implies that

$$\Pr\left[\begin{array}{c} (pk, sk) \leftarrow \mathsf{Gen}(1^k); \\ (\boldsymbol{f}, st_{\mathcal{A}}) \leftarrow \mathcal{A}^{\mathsf{Encode}_{sk}(\cdot)}(pk); \\ (\boldsymbol{f}', st) \leftarrow \mathsf{Encode}_{sk}(\boldsymbol{f}); \\ ((\boldsymbol{c}, \pi), \boldsymbol{f}^*) \leftarrow \mathcal{K}^{\mathcal{A}(st_{\mathcal{A}}, \boldsymbol{f}', st, \cdot)}(pk, st) \end{array} : \mathsf{Vrfy}(pk, st, \boldsymbol{c}, \pi) = 1 \bigwedge \boldsymbol{f}^* = \mathsf{fail} \right]$$

is negligible.

Proof. We view the $\boldsymbol{c}_j = \boldsymbol{c}(K_j)$ as vectors over $\mathbb{Z}_p$, and use the fact that integer vectors $\boldsymbol{c}_1, \ldots, \boldsymbol{c}_\ell$, with entries in the range $\{0, \ldots, p-1\}$, are linearly dependent over $\mathbb{Q}$ only if they are linearly dependent over $\mathbb{Z}_p$; thus, an upper bound on the probability of the latter implies an upper bound on the probability of the former.

Define

$$\epsilon'(k) = \max_L \left\{ \Pr[K \leftarrow \{0,1\}^k : \boldsymbol{c}(K) \in L] \right\},$$

where the maximum is taken over all $(n-1)$-dimensional subspaces $L \subset \mathbb{Z}_p^n$. It is not hard to see that if F is a non-uniformly secure PRF then $\epsilon'(k) - 1/p$ is negligible. Since $1/p$ is negligible, we see that ϵ' is negligible too. Take $\epsilon = 2\epsilon'$. We show that if $p^* > \epsilon$ then, conditioned on the extraction procedure being run, the probability that it outputs fail is negligible.

First, observe that the probability that $\mathcal{K}$ times out by virtue of running for 2^k steps is negligible (this follows from the fact that the expected running time of $\mathcal{K}$ is polynomial). Next, fix any j and consider step 3. The number of challenges that are good is exactly $p^* \cdot 2^k$, and the number of challenges K_j for which $\boldsymbol{c}(K_j)$ lies in $\mathsf{span}(\mathsf{Good}_{\boldsymbol{c}})$ (which has dimension at most $n-1$) is at most $\epsilon' \cdot 2^k < p^* \cdot 2^k/2$. Thus, the probability that a random K_j is both good and does not lie in $\mathsf{span}(\mathsf{Good}_{\boldsymbol{c}})$ is at least $p^*/2$. If $\tilde{p^*}$ is within a factor of 2 of p^*, which occurs with all but negligible probability, then $\mathcal{K}$ finds such a K_j within $k^2/\tilde{p^*}$ steps with all but negligible probability; a union bound over all values of $j \in [n]$ then shows that it fails in some iteration with only negligible probability. This completes the proof.

Finally, we show that the probability that the extraction procedure outputs an incorrect file is negligible. In conjunction with the previous claims, this completes the proof that $\mathcal{K}$ satisfies Definition 6.

Claim. For any PPT adversary $\mathcal{A}$, the following is negligible:

$$\Pr\left[\begin{array}{c} (pk, sk) \leftarrow \mathsf{Gen}(1^k); \\ (\boldsymbol{f}, st_{\mathcal{A}}) \leftarrow \mathcal{A}^{\mathsf{Encode}_{sk}(\cdot)}(pk); \\ (\boldsymbol{f}', st) \leftarrow \mathsf{Encode}_{sk}(\boldsymbol{f}); \\ ((\boldsymbol{c}, \pi), \boldsymbol{f}^*) \leftarrow \mathcal{K}^{\mathcal{A}(st_{\mathcal{A}}, \boldsymbol{f}', st, \cdot)}(pk, st) \end{array} : \begin{array}{c} \mathsf{Vrfy}(pk, st, \boldsymbol{c}, \pi) = 1 \\ \bigwedge \boldsymbol{f}^* \notin \{\mathsf{fail}, \boldsymbol{f}\} \end{array}\right].$$

Proof. The event in question can only occur if, at the end of the extraction procedure, there exists $\boldsymbol{c} \in \mathsf{Good}_{\boldsymbol{c}}$, with $\boldsymbol{c} = \boldsymbol{c}(K)$, for which $\mathcal{A}(K)$ outputs (μ, τ) such that $\mathsf{Vrfy}(pk, st, K, (\mu, \tau)) = 1$ yet $\mu \neq \sum_i c_i f_i$. But this exactly means that $\mathcal{A}$ has violated the assumed unforgeability of Λ. Since $\mathcal{K}$ runs in expected polynomial-time, it follows by a standard argument that this occurs with only negligible probability.

This concludes the proof of Theorem 2.

5 A Concrete Instantiation Based on Factoring

In this section we describe a homomorphic variant of the identification protocol of Shoup [15], whose security is based on the hardness of factoring. Together with the transformations described in the previous sections, this yields a factoring-based PoS in the random oracle model.

Protocol Σ_{Shoup}, described in Figure 3, relies on a Blum modulus generator $\mathsf{Gen}_{\mathsf{Blum}}$ that takes as input a security parameter 1^k and outputs a tuple (N, p, q) such that $N = p \cdot q$ where p and q are k-bit primes with $p = q = 3 \bmod 4$. We denote by $\mathcal{QR}_N$ the set of quadratic residues modulo N, and by $\mathcal{J}_N^{+1}$ the elements of $\mathbb{Z}_N^*$ with Jacobi symbol $+1$. We use the following standard facts regarding Blum integers: (1) given $x \in \mathbb{Z}_N^*$ it can be efficiently decided whether $x \in \mathcal{J}_N^{+1}$; (2) if $x \in \mathcal{J}_N^{+1}$, then exactly one of x or $-x$ is in $\mathcal{QR}_N$; (3) every $x \in \mathcal{QR}_N$ has four square roots, exactly one of which is itself in $\mathcal{QR}_N$.

Define homomorphic identification protocol Σ_{Shoup} as follows:

- $\mathsf{Setup}(1^k)$: Generate $(N,p,q) \leftarrow \mathsf{Gen}_{\mathsf{Blum}}(1^k)$. Choose $y \leftarrow \mathcal{QR}_N$, and output $pk := (N,y)$ and $sk := (p,q)$.
- $\mathsf{Comm}(pk;r)$: View r as an element of $\mathcal{J}_N^{+1}$ and output $\alpha := r$.
- $\mathsf{Resp}(pk,sk,r,\beta)$: Let $\beta \in \mathbb{Z}_{2^k}$ (which defines the challenge space). Output γ, a random 2^{3k}th root of $\pm r \cdot y^\beta \bmod N$ (where the sign is chosen to ensure that a square root exists).
- $\mathsf{Vrfy}(pk,\alpha,\beta,\gamma)$: Output 1 iff $\gamma^{2^{3k}} \stackrel{?}{=} \pm\alpha \cdot y^\beta \bmod N$ and $\beta < 2^{3k}$.

$\mathsf{Combine}_1$ and $\mathsf{Combine}_3$ are defined as follows:

- Let $\boldsymbol{c} \in \mathbb{Z}_{2^k}^n$ and $\boldsymbol{\alpha} \in \mathbb{Z}_N^n$. Then $\mathsf{Combine}_1(\boldsymbol{c},\boldsymbol{\alpha}) = \prod_{i=1}^n \alpha_i^{c_i} \bmod N$.
- Let $\boldsymbol{c} \in \mathbb{Z}_{2^k}^n$ and $\boldsymbol{\gamma} \in \mathbb{Z}_N^n$. Then $\mathsf{Combine}_3(\boldsymbol{c},\boldsymbol{\gamma}) = \prod_{i=1}^n \gamma_i^{c_i} \bmod N$.

Fig. 3. A homomorphic identification protocol based on factoring

Correctness of Σ_{Shoup} as a stand-alone identification protocol is immediate. Let us verify that it is homomorphic. Fix public key (N,y), challenge vector $\boldsymbol{c} \in \mathbb{Z}_{2^k}^n$, and $\{(\alpha_i,\beta_i,\gamma_i)\}_{1\le i\le n}$ such that $\gamma_i^{2^{3k}} = \pm\alpha_i \cdot y^{\beta_i} \bmod N$ for all i. Then

$$\begin{aligned}
\mathsf{Combine}_3(\boldsymbol{c},\boldsymbol{\gamma})^{2^{3k}} &= \left(\prod_{i=1}^n \gamma_i^{c_i}\right)^{2^{3k}} \bmod N \\
&= \prod_{i=1}^n \left(\gamma_i^{2^{3k}}\right)^{c_i} \bmod N \\
&= \prod_{i=1}^n \left(\pm\alpha_i \cdot y^{\beta_i}\right)^{c_i} \bmod N \\
&= \pm \prod_{i=1}^n \alpha_i^{c_i} \cdot y^{\beta_i c_i} \bmod N \\
&= \pm\mathsf{Combine}_1(\boldsymbol{c},\boldsymbol{\alpha}) \cdot y^{\sum_i c_i\beta_i} \bmod N,
\end{aligned}$$

and furthermore $\sum_i c_i\beta_i < n \cdot 2^k \cdot 2^k < 2^{3k}$.

Theorem 3. *Σ_{Shoup} is an unforgeable homomorphic identification protocol if the factoring assumption holds with respect to $\mathsf{Gen}_{\mathsf{Blum}}$.*

Proof. The high-level ideas are similar to those in [15], though the proof here is a bit simpler. Given a PPT adversary $\mathcal{A}$ attacking Σ_{Shoup}, we construct a PPT algorithm $\mathcal{B}$ computing square roots modulo N output by $\mathsf{Gen}_{\mathsf{Blum}}$. This implies factorization of N in the standard way. Algorithm $\mathcal{B}$ works as follows:

- $\mathcal{B}$ is given a Blum modulus N and a random $y \in \mathcal{QR}_N$. It runs $\mathcal{A}$ on the public key $pk = (N,y)$.
- When $\mathcal{A}$ outputs $\beta' \in \mathbb{Z}_{2^k}$, then $\mathcal{B}$ chooses random $\gamma \in \mathbb{Z}_N$ and $b \in \{0,1\}$, and sets $r := \alpha := (-1)^b \cdot \gamma^{2^{3k}}/y^\beta \bmod N$. It then gives (r,γ) to $\mathcal{A}$.

- When $\mathcal{A}$ outputs an n-vector of challenges $\boldsymbol{\beta}$, then for each i algorithm $\mathcal{B}$ computes (r_i, γ_i) as in the previous step. It gives $(\boldsymbol{r}, \boldsymbol{\gamma})$ to $\mathcal{A}$.
- If $\mathcal{A}$ outputs $(\boldsymbol{c}, \mu', \gamma')$ with $\mathsf{Vrfy}(pk, \mathsf{Combine}_1(\boldsymbol{c}, \boldsymbol{\alpha}), \mu', \gamma') = 1$ but $\mu' \neq \sum_i c_i \beta_i$, then $\mathcal{B}$ computes a square root of y as described below.

Note that the simulation provided for $\mathcal{A}$ by $\mathcal{B}$ is perfect, and so $\mathcal{A}$ succeeds in the above with the same probability with which it succeeds in attacking the real-world protocol Σ_{Shoup}.

To complete the proof, we describe the final step in more detail. Define

$$\alpha^* = \mathsf{Combine}_1(\boldsymbol{c}, \boldsymbol{\alpha}), \quad \gamma^* = \mathsf{Combine}_3(\boldsymbol{c}, \boldsymbol{\gamma}), \quad \mu = \sum_i c_i \beta_i.$$

If $\mathsf{Vrfy}(pk, \alpha^*, \mu', \gamma') = 1$ but $\mu' \neq \mu$, then $(\gamma')^{2^{3k}} = \pm\alpha^* \cdot y^{\mu'} \bmod N$; furthermore, $\mathcal{B}$ also knows that $(\gamma^*)^{2^{3k}} = \pm\alpha^* \cdot y^{\mu} \bmod N$. Assume without loss of generality that $\mu > \mu'$. Since $y \in \mathcal{QR}_N$ this implies

$$(\gamma'/\gamma^*)^{2^{3k}} = y^{\mu - \mu'} \bmod N \tag{1}$$

with $\mu, \mu' < 2^{3k}$ (and so $\mu - \mu' < 2^{3k}$). Write $\mu - \mu' = f \cdot 2^t$ for $t < 3k$ and f odd. Since squaring is a permutation of $\mathcal{QR}_N$, Equation (1) implies

$$(\gamma'/\gamma^*)^{2^{3k-t}} = y^f \bmod N.$$

Using the extended Euclidean algorithm, $\mathcal{B}$ computes integers A, B such that $Af + B2^{3k-t} = 1$. Then

$$\left(\left((\gamma'/\gamma^*)^A y^B\right)^{2^{3k-t-1}}\right)^2 = \left((\gamma'/\gamma^*)^A y^B\right)^{2^{3k-t}} = y^{Af} y^{B2^{3k-t}} = y,$$

and so $\mathcal{B}$ can compute a square root of y. Since $\mathcal{B}$ computes a square root whenever $\mathcal{A}$ succeeds, the success probability of $\mathcal{A}$ must be negligible.

Acknowledgments. We are grateful to Gene Tsudik for his insightful comments and contributions during the early stages of this work.

References

1. Ateniese, G., Burns, R., Curtmola, R., Herring, J., Kissner, L., Peterson, Z., Song, D.: Provable data possession at untrusted stores. In: ACM Conference on Computer and Communications Security. ACM, New York (2007)
2. Ateniese, G., Di Pietro, R., Mancini, L.V., Tsudik, G.: Scalable and efficient provable data possession. In: Proc. 4th Intl. Conf. on Security and Privacy in Communication Netowrks (SecureComm 2008), pp. 1–10. ACM, New York (2008)
3. Bellare, M., Goldreich, O.: On defining proofs of knowledge. In: Brickell, E.F. (ed.) CRYPTO 1992. LNCS, vol. 740, pp. 390–420. Springer, Heidelberg (1993)

4. Bowers, K., Juels, A., Oprea, A.: Proofs of retrievability: Theory and implementation. Technical Report 2008/175, Cryptology ePrint Archive (2008)
5. Dodis, Y., Vadhan, S., Wichs, D.: Proofs of retrievability via hardness amplification. In: Reingold, O. (ed.) TCC 2009. LNCS, vol. 5444, pp. 109–127. Springer, Heidelberg (2009)
6. Erway, C., Papamanthou, C., Kupcu, A., Tamassia, R.: Dynamic provable data possession. In: ACM Conf. on Computer and Communications Security (to appear, 2009). Available as Cryptology ePrint Archive, Report 2008/432
7. Feige, U., Fiat, A., Shamir, A.: Zero knowledge proofs of identity. J. Cryptology 1(2), 77–94 (1988)
8. Goldreich, O., Kahan, A.: How to construct constant-round zero-knowledge proof systems for NP. J. Cryptology 9(3), 167–190 (1996)
9. Groth, J.: A verifiable secret shuffle of homomorphic encryptions. Technical Report 2005/246, IACR ePrint Cryptography Archive (2005)
10. Guillou, L., Quisquater, J.-J.: A practical zero-knowledge protocol fitted to security microprocessor minimizing both transmission and memory. In: Günther, C.G. (ed.) EUROCRYPT 1988. LNCS, vol. 330, pp. 123–128. Springer, Heidelberg (1988)
11. Juels, A., Kaliski, B.: PORs: Proofs of retrievability for large files. In: ACM Conference on Computer and Communications Security. ACM, New York (2007)
12. Lindell, Y.: Parallel coin-tossing and constant-round secure two-party computation. J. Cryptology 16(3), 143–184 (2003)
13. Naor, M., Rothblum, G.: The complexity of online memory checking. In: IEEE Symposium on Foundations of Computer Science, pp. 573–584. IEEE Computer Society, Los Alamitos (2005)
14. Shacham, H., Waters, B.: Compact proofs of retrievability. In: Pieprzyk, J. (ed.) ASIACRYPT 2008. LNCS, vol. 5350, pp. 90–107. Springer, Heidelberg (2008), Full version `http://eprint.iacr.org`
15. Shoup, V.: On the security of a practical identification scheme. J. Cryptology 12(4), 247–260 (1999)