Efficient Authentication from PRG, Revisited

Ivan Damgård
Aarhus University

Sunoo Park
MIT

Abstract

We propose a new approach to the construction of secret-key authentication protocols from pseudorandom generators (PRG). Our authentication protocols require only two messages, have perfect completeness, and achieve the strongest meaningful security notion (man-in-the-middle security). Finally, if the PRG can be computed in poly-logarithmic depth, the authentication protocol also requires only poly-logarithmic depth computation. To the best of our knowledge, this construction is the first to have all these properties simultaneously. We achieve this at the cost of having the prover (but not the verifier) keep a small amount of state. Very efficient PRGs that can be computed in small depth can be constructed, for instance, based on the Learning Parity with Noise (LPN) problem, and our protocol is in several respects an attractive alternative even to protocols that are derived directly from LPN. A variant of our construction is secure even if the adversary is able to reset the prover.

1 Introduction

Secret-key authentication is one of most basic cryptographic tasks: a prover and a verifier share a secret key K, and the aim is to design a protocol that will allow the prover, and the prover alone, to convince the verifier that he indeed knows the key K.

The strongest security notion in the authentication literature assumes a powerful adversary who may first interact as many times as he wants with the honest prover and the verifier; after this, he is on his own and must attempt to falsely convince the verifier that he knows the secret key. If no efficient adversary can win this game, we say the protocol is man-in-the-middle (MIM) secure. An even stronger variant of this notion is *concurrent* MIM security, in which the adversary may furthermore have several concurrent sessions with (different incarnations of) the prover and the verifier, in the first phase.

In principle, one could say the authentication problem was already solved a long time ago [GGM86]. Suppose we are given a pseudorandom function family (PRF) $\mathcal{F}$ whose functions f_K have a key K and a sufficiently large input size. The crucial property of a PRF is that even an adversary who gets to choose the input (but does not know the key) cannot distinguish the output of f_K from random. Now we can simply let the verifier send a random input x, and have the prover respond with $f_K(x)$. This simple protocol is already MIM secure.

However, from the perspective of efficiency, the simple PRF solution is problematic. In [GGM86], it is shown how to construct a PRF from a pseudorandom generator (PRG), which is a much simpler primitive that expands a short key into a random-looking longer output. However, their construction yields a PRF that requires a linear depth circuit to compute[1]. Moreover, practical constructions of block-cipher-based PRFs also seem to require many sequential rounds to be secure. So it is a natural theoretical goal to build authentication protocols that require only small-depth computation, and this is also becoming ever more practically relevant as application scenarios emerge where efficient "lightweight" authentication protocols are desired: for instance, where the prover may be a low-cost RFID tag or smartcard.

[1]For authentication, it is actually sufficient to have an unpredictable function, a "MAC", rather than a PRF; however, we do not know of more efficient constructions of this primitive, either.

Now, there are several very efficient and low-depth constructions of PRGs from specific problems, such as Learning Parity with Noise (LPN); and furthermore, the existence of low-depth PRGs is known to follow from much more general assumptions [App13]. Thus, it is compelling to try to design authentication directly from weaker pseudorandom primitives such as PRGs.

This direction has been explored in [DKPW12], which constructs a three-round protocol based on any weak PRF. A weak PRF is a relaxed notion of PRF where function outputs are only required to look random for uniformly random (rather than adversarial) inputs. The protocol of [DKPW12] achieves active security (a weaker notion than MIM security). Subsequently, [LM13] proposed a three-round protocol based on any weak PRF, which is secure against (sequential) MIM attacks. In addition, they give a variant three-round protocol that can be built from any *randomized* weak PRF, a yet slightly weaker primitive. However, these protocols are not concurrent MIM secure (in fact, [LM13] outlines an attack). Moreover, even a randomized weak PRF seems to be a significantly stronger primitive than a PRG.

Finally, an obvious alternative approach to authentication is to build protocols directly from a concrete problem. Much work in this direction has been based on the LPN problem, which can be briefly stated as follows: given polynomially many samples of the form $(\mathbf{a}, \langle \mathbf{a}, \mathbf{s} \rangle + e)$, where $\mathbf{a}$ is a random n-bit vector, $\mathbf{s}$ is a secret n-bit vector fixed across all samples, and e is a bit which is 1 with probability $\tau < 1/2$, can we discover the secret $\mathbf{s}$ (or at least, distinguish the samples from random)? Authentication from LPN has been studied in a series of papers, including [HB01; JW05; KSS10; Hey+12]. The latest in this line of work [Kil+11; DKPW12] have proposed two-round MIM secure protocols. These protocols require only small-depth computation, but they suffer from a rather large (quadratic) key size, and moreover they are not perfectly complete: the verifier may reject the honest prover with non-zero probability. In order to keep this error probability low, the protocol's communication (and hence also computational) complexity must depend on the LPN noise parameter τ, and in fact the complexity grows quite dramatically as τ approaches $1/2$. Thus, these protocols incur a rather high price if we want to use a weaker variant of the LPN assumption.

Our contribution. We propose new two-round authentication protocols that can be based on black-box access to any PRG, using two different approaches.

The first protocol has perfect completeness and is concurrent MIM secure for any polynomial number of concurrent sessions, as long as this polynomial is fixed when the system is set up. Moreover, if the PRG can be computed in poly-logarithmic depth, then the computation needed in the protocol is poly-logarithmic depth as well. To the best of our knowledge, our protocol is the first to have all these properties simultaneously. We achieve this by having the prover (but not the verifier) keep a small amount of state (a counter). The total computation needed for the protocol is $O(\log(t))$ where t is the number of times the protocol is used[2].

We then show a particularly efficient instantiation of this protocol based on LPN, and since we have perfect completeness (and thus avoid the bad dependence on τ mentioned above), this offers an attractive alternative to known protocols based directly on LPN. In particular, unlike previous protocols, our protocol has the property that as we move τ towards $1/2$ to weaken the underlying LPN assumption, we pay only in terms of computational complexity: our communication complexity stays unchanged.

The basic idea of our protocol is that the verifier sends a n-bit random challenge a to the prover, who responds with a MAC on a computed from a secret key that he shares with the verifier. To get the efficiency and low depth that we are after, we will use a well-known unconditionally secure MAC of the form $as+e$ where (s, e) is the key and the computation is in the field with 2^n elements.

The problem with this is that while s can be reused over several executions, e cannot, it must be a fresh random value every time, or the MAC is not secure. An obvious solution is to choose e pseudorandomly using another shared key. Computing it as $e = f_K(a)$ where f is PRF may seem

[2] It should be noted that this refers to the time spent in normal operation between the honest prover and verifier. An adversary can force the verifier (but not the prover) to spend more time, but we do not view this as a significant problem: in practice, an adversary could always waste the verifier's time by doing a standard denial of service attack.

natural, but this would be pointless: if we already have a PRF, the simpler protocol mentioned above might as well be used, and besides we do not have low-depth constructions of PRF from PRG. Our idea is to instead have the prover keep a counter i that is incremented for each execution, and compute e as $e = f_K(i)$. The point is that now the prover does not need to compute the PRF on arbitrary inputs, but only on consecutive small values $i = 1, 2, \ldots$. We use a variant of the GGM construction to build a PRF that can be computed in small depth for exactly such inputs, while still being secure and no less efficient than GGM for adversarially chosen inputs.

This basic protocol satisfies (sequential) MIM security. We then achieve concurrent MIM security by an additional technique based on universal hashing.

Our second contribution is another two-round authentication protocol with properties that are incomparable to the first protocol, as detailed later in this section. Here, we assume an efficiently computable function g such that for random strings a, x, the pair $(a, g(a, x))$ is indistinguishable from random, and is longer than the input string (a, x). We call this a PRG with public randomizer. The existence of such functions follows trivially from existence of a standard PRG G, since we can set $g(a, x) = G(x)$; however, for certain problems such as LPN or subset sum, we can instantiate g in a more direct way that can lead to a better security guarantee for the protocol.

From a function g as described above, we construct a "bounded" PRF which can be computed in poly-logarithmic depth if the same is true for g. When setting up keys for the PRF, we also set a parameter ℓ, and the PRF will then be secure if g is a PRG with public randomizer, provided that the adversary asks at most ℓ queries (ℓ can be any polynomial in the security parameter). If more than ℓ queries are asked, this same PRF is still secure, but now under a stronger assumption – namely, that samples of the following form are indistinguishable from random: $(a_1, g(a_1, x_1)), (a_2, g(a_2, x_2)), \ldots$, where the a_i are random, but the x_i are chosen from an ℓ-wise independent distribution, and the number of samples is polynomial.

Our protocol based on this PRF is (concurrent) MIM secure assuming only that g is a PRG with public randomizer, so it can be instantiated from any PRG (we do not need the stronger assumption for this). The local computation for this protocol is somewhat more complicated than for the first one, and it can be expected to be less efficient in practice; however, it still requires only poly-logarithmic depth computation as long as g is computable in poly-log depth. Furthermore, the run-time of our second protocol grows more slowly with the number of protocol executions t: it is $O(\log(\lceil t/\ell \rceil))$.

A different type of distinction between the two protocols emerges if one considers *resetting attacks* on the prover: while we believe that asking (only) the prover to keep a small amount of state is reasonable, it is nevertheless natural to ask what happens if the adversary is able to reset the prover. The first protocol is insecure under such an attack, while the second is secure, but only under the stronger assumption mentioned above.

2 Preliminaries

Notation. For a finite set B, we will write $b \leftarrow B$ to denote that b is drawn uniformly randomly from B. For $n \in \mathbb{N}$, $[n]$ denotes the set $\{1, 2, \ldots, n\}$. The relation $\overset{s}{\approx}$ between distributions denotes statistical indistinguishability, and $\overset{c}{\approx}$ denotes computational indistinguishability. $\mathsf{negl}(n)$ denotes a negligible function in parameter n, and $\mathsf{poly}(n)$ denotes a polynomial function. An *efficient* algorithm is one which runs in probabilistic polynomial time (PPT).

2.1 Authentication protocols

A *authentication protocol* is an interactive two-party protocol $(\mathcal{P}, \mathcal{V})$ between a prover $\mathcal{P}$ and a verifier $\mathcal{V}$: these may be respectively thought of as a (lightweight) *tag*, and a *reader* to which the tag is identifying itself. Both parties are PPT, and hold a shared secret s generated according to some generation algorithm $\mathsf{Gen}(1^\kappa)$ (where κ denotes the security parameter) in an initial phase.

After an execution of the protocol, the verifier $\mathcal{V}$ outputs either accept or reject – this is also called the *output* of the protocol execution.

In this work we consider *prover-stateful protocols* where the prover also maintains some (small amount of) state between protocol executions.

Definition 2.1 (Completeness)**.** *The* completeness error *of a protocol is defined to be*

$$\Pr_{s \leftarrow \mathsf{Gen}(1^\kappa)}[(\mathcal{P}, \mathcal{V})(s) = \text{accept}].$$

A protocol is complete *if its completeness error is negligible in a security parameter. It is* perfectly complete *if its completeness error is zero.*

Common definitions of security for authentication protocols are given below. The security definitions are presented in order of increasing strength, and the stronger security notions subsume the weaker ones.

Definition 2.2 (Passive security)**.** *An authentication protocol* $(\mathcal{P}, \mathcal{V})$ *is* secure against passive attacks *if for any secret* $s \leftarrow \mathsf{Gen}(1^\kappa)$*, for any* PPT *adversary* $\mathcal{A}$ *which has access to arbitrarily polynomially many transcripts of honest protocol executions (for secret* s*), it holds that*

$$\Pr[(\mathcal{A}, \mathcal{V})(s) = \text{accept}] \leq \mathsf{negl}(\kappa).$$

Definition 2.3 (Active security)**.** *An authentication protocol* $(\mathcal{P}, \mathcal{V})$ *is* secure against active attacks *if for any secret* $s \leftarrow \mathsf{Gen}(1^\kappa)$*, for any* PPT *adversary* $\mathcal{A}$ *which first can interact arbitrarily polynomially many times with an honest prover* $\mathcal{P}$ *(but cannot reset the prover's state), and then afterward (now, without access to* $\mathcal{P}$*) interacts once with an honest verifier* $\mathcal{V}$*, it holds that*

$$\Pr[(\mathcal{A}, \mathcal{V})(s) = \text{accept}] \leq \mathsf{negl}(\kappa).$$

Definition 2.4 (Sequential man-in-the-middle (MIM) security)**.** *An authentication protocol* $(\mathcal{P}, \mathcal{V})$ *is* secure against sequential man-in-the-middle attacks *if for any* PPT *adversary* $\mathcal{A}$ *which first can interact arbitrarily polynomially many times with an honest prover* $\mathcal{P}$ *and/or an honest verifier* $\mathcal{V}$ *(but cannot reset the prover's state), and then afterward (now, without access to* $\mathcal{P}, \mathcal{V}$*) interacts once with an honest verifier* $\mathcal{V}'$*, it holds that*

$$\Pr[(\mathcal{A}, \mathcal{V}')(s) = \text{accept}] \leq \mathsf{negl}(\kappa).$$

In this setting, the adversary learns the accept/reject *decisions made by the verifier(s).*

Definition 2.5 (Concurrent man-in-the-middle (MIM) security)**.** *An authentication protocol* $(\mathcal{P}, \mathcal{V})$ *is* secure against concurrent man-in-the-middle attacks *if for any* PPT *adversary* $\mathcal{A}$ *which first can interact arbitrarily polynomially many times with polynomially many honest provers* $\mathcal{P}_1, \ldots, \mathcal{P}_k$ *and/or honest verifiers* $\mathcal{V}_1, \ldots, \mathcal{V}_k$ *(but cannot reset the provers' states), and then afterward (now, without access to the* $\mathcal{P}_i, \mathcal{V}_i$*) interacts once with an honest verifier* $\mathcal{V}'$*, it holds that*

$$\Pr[(\mathcal{A}, \mathcal{V}')(s) = \text{accept}] \leq \mathsf{negl}(\kappa).$$

As above, the adversary learns the accept/reject *decisions made by the verifier(s).*

Variants of security definitions. The definition of concurrent MIM security can be relaxed slightly to give a notion of bounded-concurrent security. We write "ℓ-concurrent MIM security" to denote security against an adversary who has concurrent access to up to ℓ honest provers (and verifiers), but no more. Similarly, one can define concurrent and bounded-concurrent active security. Finally, one can consider strengthening any of the definitions above by allowing the adversary the power to reset the prover; this yields an "ultimate" security notion of *reset-safe concurrent man-in-the-middle security.*

2.2 Pseudorandom primitives

We recall below the definitions of pseudorandom generators (PRGs) and pseudorandom function families (PRFs).

Definition 2.6 (Pseudorandom generator)**.** *Let $G : \{0,1\}^n \to \{0,1\}^{m(n)}$ be a deterministic polynomial-time algorithm. G is a* pseudorandom generator *(PRG) if $m(n) > n$ and for any efficient distinguisher D that outputs a single bit, it holds that $|\Pr[\mathsf{D}(r) = 1] - \Pr[\mathsf{D}(G(s)) = 1]| \leq \mathsf{negl}(n)$, where $r \leftarrow \{0,1\}^{m(n)}, s \leftarrow \{0,1\}^n$ are chosen uniformly at random, and the probabilities are taken over r, s, and the random coins of D.*

It is well known that any pseudorandom generator implies pseudorandom generation with any polynomial expansion factor $m(n)$, by applying the PRG to its own output repeatedly.

Definition 2.7 (Pseudorandom function (PRF))**.** *Let $\mathcal{F} = \{F_K\}$ be family of deterministic polynomial-time keyed algorithms mapping n bits to m bits. $\mathcal{F}$ is a* pseudorandom function family (PRF)*, if for any efficient distinguisher D that outputs a single bit, it holds that $|Pr[\mathsf{D}^{F_K} = 1] - \Pr[\mathsf{D}^{\mathcal{R}_{n\to m}} = 1]| \leq \mathsf{negl}(n)$, where $\mathcal{R}_{n\to m}$ is a random oracle mapping n bits to m bits, and the probabilities are taken over the random coins of D and the key K which is randomly chosen.*

A variant notion that we consider is that of a "b-bounded PRF", which is exactly like a PRF except that security is only guaranteed if the adversary makes no more than b oracle queries.

Weak pseudorandom functions (WPRFs) [NR99] are functions whose outputs appear random for *random* inputs, as opposed to *arbitrary* inputs as in the PRF definition. Since WPRFs can be more efficient than PRFs, the use of WPRFs is favoured in applications where they suffice.

Definition 2.8 (Weak pseudorandom function (WPRF))**.** *Let $\mathcal{F} = \{F_K\}$ be family of deterministic polynomial-time keyed algorithms mapping n bits to m bits. Let $\mathcal{O}^f$ be an oracle that takes no input but when queried returns $(x, f(x))$ where $x \leftarrow \{0,1\}^n$ is chosen randomly. $\mathcal{F}$ is a* weak pseudorandom function family (WPRF)*, if for any efficient distinguisher D that outputs a single bit, it holds that $|Pr[\mathsf{D}^{\mathcal{O}^{F_K}} = 1] - \Pr[\mathsf{D}^{\mathcal{O}^{\mathcal{R}_{n\to m}}} = 1]| \leq \mathsf{negl}(n)$, and the probabilities are taken over the random coins of D and the key K which is chosen uniformly at random.*

3 Overview of protocols

In this section we give an overview of known protocols and techniques for efficient authentication. Two main approaches in this field have been to construct protocols in a black-box way from pseudorandom primitives, and to propose protocols based on concrete hardness assumptions such as LPN. In the latter direction, LPN and its variants have been particularly favoured in the literature as they involve fast computations that are well-suited for computationally limited devices[3].

3.1 Efficient authentication from pseudorandom primitives

Given a PRF, it is possible to construct a simple two-round protocol achieving fully concurrent MIM security. In fact, this construction does not require a full-blown PRF: a message authentication code (MAC) would suffice. The catch is that in such protocols, the prover must evaluate the PRF, and known PRF constructions are relatively inefficient (certainly too slow to be deployed on lightweight tag devices). Furthermore, known MAC constructions tend to be based on PRFs.

Thus, it is natural to ask: can we construct *efficient* authentication protocols from weaker pseudorandom primitives[4]? While earlier works in the authentication literature focused on *ad*

[3] Or even, as suggested in the original work of [HB01] which introduced LPN-based authentication, potentially suitable for computation by humans!

[4] Given a PRG or a weak PRF, we can build an authentication protocol *through* a PRF via the classic tree construction of [GGM86], but this will again be inefficient. It is not known in general how to more efficiently obtain a PRF based on a PRG or WPRF.

hoc constructions based on specific assumptions (some of which are discussed in Section 3.2), this more general approach has gained attention in recent years, and seems particularly promising for achieving strong MIM security. [DKPW12] constructed a three-round actively secure protocol based on any weak PRF. Building upon this, [LM13] gave a three-round protocol from any weak PRF, which is secure against sequential MIM attacks. In addition, [LM13] proposes a similar three-round protocol that can be built from any *randomized* weak PRF, a yet slightly weaker primitive than WPRF. However, their protocols are not concurrent MIM secure: indeed, they outline a possible attack in the case of just two concurrent provers.

3.2 Efficient authentication from LPN and variants

Notation. We denote vectors by lower-case letters, and matrices by upper-case letters. Arithmetic operations on n-bit vectors are taken over the field of 2^n elements. When adding an n-bit vector to an m-bit vector where $n \neq m$, we assume the shorter one to be zero-padded to match the length of the longer one. Let Ber_τ denote the Bernoulli distribution with parameter τ, and let Ber_τ^n denote the distribution of vectors in $\{0,1\}^n$ where each bit is independently distributed as Ber_τ.

3.2.1 The LPN problem

The decisional LPN problem is that of distinguishing from random a set of samples, each of the form $(a, \langle a, s\rangle + e)$, where $a \in \{0,1\}^n$ is uniformly random, $e \leftarrow \mathsf{Ber}_\tau$, and $s \in \{0,1\}^n$ is a randomly chosen secret that is fixed over all samples.

Definition 3.1 (Decisional LPN problem [BFKL94])**.** *Take parameters $n \in \mathbb{N}$ and $\tau \in \mathbb{R}$ with $0 < \tau \leq \frac{1}{2}$ (the noise rate). A distinguisher D is said to (q, t, ε)-solve the decisional $\mathsf{LPN}_{n,\tau}$ problem if*

$$\left| \Pr_{s,A,e}[\mathsf{D}(A, A \cdot s + e) = 1] - \Pr_{r,A}[\mathsf{D}(A, r) = 1] \right| \geq \varepsilon$$

where $s \leftarrow \{0,1\}^n$, $A \leftarrow \{0,1\}^{q \times n}$, and $r \leftarrow \{0,1\}^q$ are uniformly random and $e \leftarrow \mathsf{Ber}_\tau^q$, and the distinguisher runs in time at most t.

In the search version of the problem, the goal is instead to find the secret vector s. The decisional and search versions of the LPN problem are polynomially equivalent [KSS10].

3.2.2 LPN-based authentication protocols

The first and simplest authentication scheme based on LPN was the HB scheme [HB01], illustrated in Protocol 1. HB is secure against passive attacks (but easily breakable by active attacks). Subsequently, [JW05] gave an actively secure[5] variant protocol called HB$^+$ with an additional round (which requires the prover to keep state between rounds). Then, [Kil+11] proposed the first two-round actively secure protocol, whose security is based on the Subspace LPN problem[6].

The preceding protocols are all vulnerable to man-in-the-middle attacks. [Kil+11] shows how to generate a MAC based on their aforementioned actively secure authentication protocol, and this yields the first somewhat efficient MIM secure protocol based on LPN (using two rounds). More recently, [DKPW12] gave a more efficient MIM secure (three-round) variant protocol using more efficient MACs that make use of pairwise independent hashing. Unfortunately, both of these constructions suffer from large (quadratic) MAC key-sizes.

There is also interest in basing protocols on variant LPN assumptions such as ring-LPN and Toeplitz-LPN, which impose some additional structure on the public matrix A, and consequently

[5] In fact, it was subsequently shown that HB$^+$ is secure even against *concurrent active* attacks where the adversary may interact with multiple provers concurrently [KSS10].

[6] Subspace LPN is a variant of the LPN problem that has been shown to be almost as secure as standard LPN, under certain conditions [PP03].

Public parameters. Security parameter $\kappa \in \mathbb{Z}$, $n \in \mathbb{Z}$ with $n = \mathsf{poly}(\kappa)$, noise rate $\tau \in (0, \frac{1}{2})$, threshold $\tau' \in (\tau, \frac{1}{2})$.

Key generation. $\mathsf{Gen}(1^\kappa)$ samples secret key $s \leftarrow \mathbb{Z}_2^n$.

$\mathcal{P}(s)$		$\mathcal{V}(s)$
	$\xleftarrow{A}$	$A \leftarrow \mathbb{Z}_2^{n \times m}$
$e \leftarrow \mathsf{Ber}_\tau^n$ $z := As + e$	$\xrightarrow{z}$	accept iff $\|\|z + As\|\|_1 < \tau' \cdot n$

Protocol 1: The HB authentication protocol

allow yet faster operations as well as smaller key sizes[7]. The preceding LPN-based protocols can all be straightforwardly adapted to the ring- or Toeplitz-LPN settings (preserving security properties, but with respect to the variant assumption rather than the LPN assumption), except for the MAC-based [Kil+11] and [DKPW12] which depend on particular algebraic properties of the LPN problem. Further, there have been protocols designed particularly with variant assumptions in mind: notably, [Hey+12] which depends on ring-LPN.

4 PRG-based authentication via "one-time" MACs

In this section, our approach is to build authentication protocols from very efficient MACs. It was observed in earlier sections that known MAC constructions are not very efficient as they are based on PRFs – however, this is only true of *computationally* secure MACs. In contrast, there are *unconditionally* secure MACs that can be computed very efficiently – but these are only secure for one-time use, so have not thus far been considered suitable for authentication protocols.

We focus on the following simple and unconditionally secure MAC: for a message $a \in \{0,1\}^n$, the MAC on the message is $a \cdot s + e$, where $(s, e) \in \{0,1\}^n \times \{0,1\}^n$ is the secret key (which is chosen uniformly at random). MACs of this form are well known, and it is also known that although a key for an unconditionally secure MAC can usually be used only once, in this case the multiplier (s) can be reused provided that e is freshly chosen for each message (see e.g. [BDOZ11]). In our protocols, we consider s to be the secret key, and generate e pseudorandomly per execution. Note that the man-in-the-middle security of our protocol does *not* follow from MAC security, however: the standard security notion for MACs simply requires that an adversary who observes a message and a valid MAC cannot produce a different message and valid MAC. We consider a more complicated game where the adversary interacts with prover and verifier concurrently.

We initially construct a slightly simpler protocol that is secure against sequential MIM attacks, then propose a variant (still two-round) protocol that is furthermore secure against ℓ-concurrent MIM attacks at very little additional overhead, where ℓ can be any polynomial in the security parameter but must be fixed in advance.

Notation. Since a PRG G can be used to build a PRG of any stretch, we write $G_{n \to m}$ to denote the PRG based on G which maps n bits to m bits. We write $G_{n \to m}(r)^{[i,j]}$ to denote the substring of the PRG output $G_{n \to m}(r) \in \{0,1\}^m$ ranging from the i^{th} bit to the j^{th} bit, inclusive.

[7] For example, in ring-LPN, vectors in $\mathbb{Z}_2^n$ are interpreted as elements of a polynomial ring $R = \mathbb{F}_2[X]/(g)$ for some $g \in \mathbb{F}_2[X]$ of degree $n-1$ whose coefficients correspond to the bits of the vector. As a result, the relatively expensive matrix multiplication in computing $As+e$ can be replaced by a much faster ring multiplication in computing $a \cdot s + e$.

4.1 Pseudorandom look-up function

To begin with, as a building block for our authentication protocols, we construct a logarithmic-depth "look-up function" for efficient retrieval of pseudorandom values using the PRG, and show that the look-up function is a PRF. Note that this technique may be of independent interest towards generic constructions of low-depth PRFs.

Given a PRG G taking an n-bit input, our goal is to generate a series of pseudorandom values $r_1, \ldots, r_t \in \{0,1\}^m$ for some $m \in \mathbb{N}$, such that each r_i can be looked up in logarithmic time. This is achieved using the tree structure below.

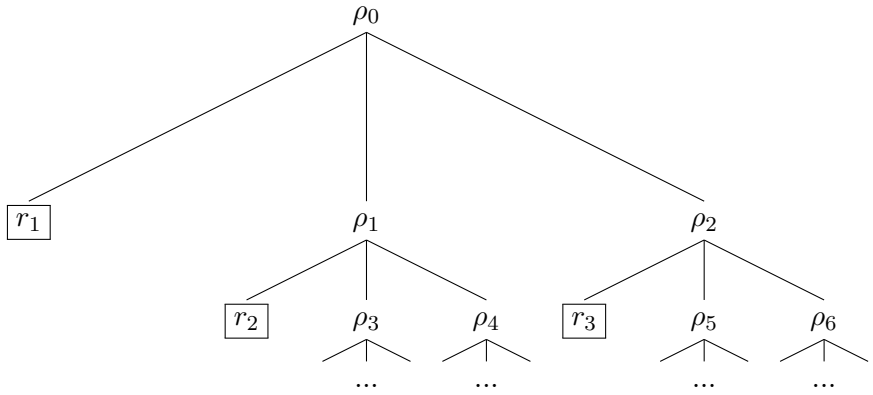

Figure 1: Tree illustrating efficient look-up of pseudorandom values

In Figure 1, $\rho_0 \in \{0,1\}^n$ is the original (random) input to the PRG $G_{n \to m+2n}$. The $\rho_i \in \{0,1\}^n$ are values which are subsequently pseudorandomly generated, which are used again as input to the PRG to produce more pseudorandom values: in particular, if ρ_i is a child of ρ_j in the tree, then $\rho_i = G_{n \to m+2n}(\rho_j)^{[m+1,m+n]}$ if i is even, and $\rho_i = G_{n \to m+2n}(\rho_j)^{[m+n+1,m+2n]}$ if i is odd. The boxed nodes $r_i \in \{0,1\}^m$ are leaves that represent the output pseudorandom values which we want to look up, and they are generated by $r_i = G_n(\rho_j)^{[1,m]}$ where ρ_j is the parent node of r_i.

Let $\mathsf{lookup}^G_{n,m}(\rho_0, i) \in \{0,1\}^m$ denote the i^{th} output value, $r_i \in \{0,1\}^m$, obtained using the above tree method. It is clear that for any i of polynomial size, the number of PRG evaluations required to look up r_i is logarithmic. This gives rise to a PRF family with logarithmic-depth evaluations, as proven in Theorem 4.2 below. Before proving that the look-up function is a PRF, we give a simple supporting lemma.

Lemma 4.1. *Let $G : \{0,1\}^n \to \{0,1\}^m$ be a PRG. Then for any polynomial $q = q(n)$, it holds that there is no efficient distinguisher D for which it holds that*

$$|\Pr[\mathsf{D}((r_1, \ldots, r_q)) = 1] - \Pr[\mathsf{D}((G(s_1), \ldots, G(s_q))) = 1]| \geq \mathsf{negl}(n)$$

for all negligible functions negl, where $r_1, \ldots, r_q \leftarrow \{0,1\}^{m(n)}$ and $s_1, \ldots, s_q \leftarrow \{0,1\}^n$.

Proof. Suppose, for contradiction, that there is a distinguisher $\widehat{\mathsf{D}}$ for which

$$\left|\Pr\left[\widehat{\mathsf{D}}((r_1, \ldots, r_q)) = 1\right] - \Pr\left[\widehat{\mathsf{D}}((G(s_1), \ldots, G(s_q))) = 1\right]\right| \geq 1/P(n)$$

where P is a polynomial.

For $i \in [q]$, define tup_i to be the distribution of tuples whose first i elements are uniformly random in $\{0,1\}^m$ and whose remaining elements are sampled as $G(s_{i+1}), \ldots, G(s_q)$ for $s_{i+1}, \ldots, s_q \leftarrow \{0,1\}^n$. Let $p_i = \Pr[\widehat{\mathsf{D}}(\mathsf{tup}_i) = 1]$ denote the probability that $\widehat{\mathsf{D}}$ outputs 1 on input from tup_i.

By our supposition, we know that $|p_0 - p_q| \geq P(n)$. Since $p_0 - p_q = \sum_{i \in [q]}(p_{i-1} - p_i)$, it follows that there must exist some $i^* \in [q]$ such that $|p_{i^*-1} - p_{i^*}| \geq \frac{1}{q \cdot P(n)}$, which is non-negligible. Then there exists a distinguisher $\widehat{\mathsf{D}}'$ which can distinguish a *single* output of the PRG from random,

as follows: on input $r \in \{0,1\}^m$, $\widehat{\mathsf{D}}'$ generates a tuple t whose first $i^* - 1$ elements are uniformly random in $\{0,1\}^m$, whose $(i^*)^{th}$ element is r, and whose remaining elements are generated as $G(s_{i^*+1}), \ldots, G(s_q)$ for $s_{i+1}, \ldots, s_q \leftarrow \{0,1\}^n$. If r is truly random then $t \leftarrow \mathsf{tup}_{i^*}$; otherwise, $t \leftarrow \mathsf{tup}_{i^*-1}$. Hence, running $\widehat{\mathsf{D}}$ on input t will distinguish with non-negligible probability between these two cases. This contradicts that G is a PRG. □

Theorem 4.2. *Let G be a PRG and $n, m \in \mathbb{N}$ be positive integers with $m = \mathsf{poly}(n)$. Then the family of functions $\mathcal{F}^{(n,m)} \stackrel{\text{def}}{=} \left\{\mathsf{lookup}^G_{n,m}(\rho, \cdot)\right\}_{\rho \in \{0,1\}^n}$ is a PRF with input size n' bits and output size n bits, for any $n' = \mathsf{poly}(n)$.*

Proof. The statement to prove is that for any PRG G and random $\rho_0 \leftarrow \{0,1\}^n$, there is no efficient distinguisher D that satisfies

$$\left|\Pr\left[\mathsf{D}^{\mathsf{lookup}^G_{n,m}(\rho_0,\cdot)}() = 1\right] - \Pr\left[\mathsf{D}^{\mathcal{R}(\cdot)}() = 1\right]\right| > \mathsf{negl}(\kappa)$$

for all negligible functions negl and all sufficiently large values of the security parameter κ. In the above, $\mathcal{R}$ is a random oracle, and $\mathsf{D}^{\mathcal{O}}$ may make any polynomial number of "polynomial-depth"[8] queries to the oracle $\mathcal{O}$.

Suppose, for contradiction, that there exists such a distinguisher $\widehat{\mathsf{D}}$, for which the above inequality does not hold: that is, there is a polynomial q for which

$$\left|\Pr\left[\mathsf{D}^{\mathsf{lookup}^G_{n,m}(\rho_0,\cdot)}() = 1\right] - \Pr\left[\mathsf{D}^{\mathcal{R}(\cdot)}() = 1\right]\right| > 1/q(\kappa).$$

Let $T = T(\kappa)$ be the run-time of distinguisher $\widehat{\mathsf{D}}$, and let H_j be a stateful algorithm which is defined as follows. (Note that when referring to tree structure, the root node is considered to be at depth 0.) On input i, the algorithm H_j does the following:

- if i has already been queried previously, look up the stored tuple $(\mathsf{leaf}, i, \rho_i)$, and output ρ_i;
- else if $i < 2^{j+1}$ (that is, the i^{th} output node is at depth less than or equal to j in the tree), then choose some $\rho_i \leftarrow \mathbb{Z}_n^2$ uniformly at random, store the tuple $(\mathsf{leaf}, i, \rho_i)$ in memory, and output ρ_i;
- otherwise (that is, the i^{th} output node is at depth greater than j in the tree):
  - if there is no stored tuple of the form $(\mathsf{root}, \alpha, \rho)$, then choose some $\rho \leftarrow \mathbb{Z}_n^2$ uniformly at random, store the tuple $(\mathsf{root}, \alpha, \rho)$ in memory, and output $\mathsf{lookup}^G_{n,m}(\rho, \gamma)$;
  - otherwise, look up the stored tuple $(\mathsf{root}, \alpha, \rho)$ and output $\mathsf{lookup}^G_{n,m}(\rho, \gamma)$;

  where $\alpha = \alpha_{(i,j)}, \gamma = \gamma_{(i,j)}$ are defined by the following:

$$\begin{aligned}
\alpha_{(i,j)} &= \lfloor (i - 2^{\lfloor \log_2(i) \rfloor})/2^j \rfloor \\
\beta_{(i,j)} &= i - 2^{\lfloor \log_2(i) \rfloor} \mod 2^{\lfloor \log_2(i) \rfloor - j} \\
\gamma_{(i,j)} &= 2^{(2^{\lfloor \log_2(i) \rfloor - j})} + \beta_{(i,j)}.
\end{aligned}$$

When considering the tree representation of $\mathsf{lookup}^G_{n,m}$, the algorithms H_j can be explained in more intuitive terms as follows. For each j, the outputs of H_j behave as a random oracle up to and including depth j of the tree. Below depth j, the outputs are obtained deterministically by the $\mathsf{lookup}^G_{n,m}(\rho, \cdot)$ function, with the appropriate depth-j value ρ (which is randomly chosen) acting as the "root node" of the subtree in which the lookup is performed.

[8] More precisely, the distinguisher cannot make queries that would require a super-polynomial depth look-up in the tree structure of $\mathsf{lookup}^G_{n,m}$. This is because in this case, $\mathsf{lookup}^G_{n,m}$ would not run in polynomial time.

Observe that H_0 behaves exactly as $\mathsf{lookup}^G_{n,m}(\rho, \cdot)$ for random $\rho \leftarrow \{0,1\}^n$. Moreover, H_k behaves exactly like a random oracle in the distinguishing experiment, provided that all of $\widehat{\mathsf{D}}$'s queries can be retrieved from depth at most k. Since the input size is $n' = \mathsf{poly}(n)$ bits, there exists such a maximum depth k from which queries can be retrieved, with $k = \mathsf{poly}(n)$.

Let $p_i = \Pr[\widehat{\mathsf{D}}^{H_i}() = 1]$ denote the probability that the distinguisher outputs 1 given H_i as an oracle. By our earlier supposition, $|p_0 - p_k| > 1/q(\kappa)$. It follows that for some $k^* \in [k]$, we have $|p_{k^*-1} - p_{k^*}| > \frac{1}{k \cdot q(\kappa)}$. Such a k^* can be found in polynomial time with non-negligible probability[9].

We now construct a new distinguisher $\widehat{\mathsf{D}}_{\mathsf{PRG}}$ attacking the PRG $G_{n \to m+2n}$: specifically, we will show that the following expression is non-negligible:

$$\left| \Pr\left[\widehat{\mathsf{D}}_{\mathsf{PRG}}((r_1, \ldots, r_T)) = 1\right] - \Pr\left[\widehat{\mathsf{D}}_{\mathsf{PRG}}((G_{n \to m+2n}(s_1), \ldots, G_{n \to m+2n}(s_T))) = 1\right] \right|,$$

where $r_1, \ldots, r_T \leftarrow \{0,1\}^{m+2n}$ and $s_1, \ldots, s_T \leftarrow \{0,1\}^n$.

$\widehat{\mathsf{D}}_{\mathsf{PRG}}$ operates as follows. Given input $(\widetilde{r}_1, \ldots, \widetilde{r}_T)$, $\widehat{\mathsf{D}}_{\mathsf{PRG}}$ first determines a $k^* \in [k]$ as described above, then runs $\widehat{\mathsf{D}}$ and responds to the oracle queries of $\widehat{\mathsf{D}}$ in the following way: when $\widehat{\mathsf{D}}$ makes query i, $\widehat{\mathsf{D}}_{\mathsf{PRG}}$ responds with $\widetilde{H}_{k^*}(\widetilde{r}_1, \ldots, \widetilde{r}_T)$, where $\widetilde{H}_{k^*}$ is a variant algorithm based on H_{k^*}. $\widetilde{H}_{k^*}$ takes as input $(\widetilde{r}_1, \ldots, \widetilde{r}_T)$, and then behaves exactly like H_{k^*}, except that the values associated with nodes at depth k^* of the tree are obtained as substrings of the input values $\widetilde{r}_1, \ldots, \widetilde{r}_T$. (For a detailed formal description of $\widetilde{H}_{k^*}$, refer to Appendix A.)

By construction, it holds that if the inputs $\widetilde{r}_i$ are truly random, then $\widetilde{H}_{k^*}$ and H_{k^*} behave identically; on the other hand, if the inputs $\widetilde{r}_i$ are generated by $G_{n \to m+2n}$, then $\widetilde{H}_{k^*}$ and H_{k^*-1} behave identically. By the choice of k^*, we know that $\widehat{\mathsf{D}}$ distinguishes H_{k^*} and H_{k^*-1} with non-negligible probability. Hence, $\widehat{\mathsf{D}}_{\mathsf{PRG}}$ distinguishes with (the same) non-negligible probability between the case where the $\widetilde{r}_1, \ldots, \widetilde{r}_T$ are random and the case where they are generated by $G_{n \to m+2n}$. By Lemma 4.1, this contradicts that $G_{n \to m+2n}$ is a PRG. Therefore, our initial supposition was false: that is, there cannot exist a $\widehat{\mathsf{D}}$ which distinguishes between $\mathsf{lookup}^G_{n,m}(\rho_0, \cdot)$ and $\mathcal{R}$ with non-negligible probability. The lemma follows. □

4.2 The protocol constructions

We now present our protocol constructions which can be instantiated by any PRG.

4.2.1 Sequential man-in-the-middle secure protocol

Public parameters. PRG G, security parameter $n \in \mathbb{Z}$.
Key generation. $\mathsf{Gen}(1^n)$ samples $s, s' \leftarrow \{0,1\}^n$ and outputs secret key (s, s').
Initial state. The prover's state consists of $i \in \mathbb{N}$ initialised to 1.

$\mathcal{P}(s, s'; i)$		$\mathcal{V}(s, s')$
	$\xleftarrow{a}$	$a \leftarrow R$
$e := \mathsf{lookup}^G_{n,n}(s', i)$ $z := a \cdot s + e$ $i := i + 1$	$\xrightarrow{z,\ i}$	accept iff $z + a \cdot s = \mathsf{lookup}^G_{n,n}(s', i)$

Protocol 2: Sequential MIM secure protocol based on any PRG

[9] This can be done by running $\widehat{\mathsf{D}}$ polynomially many times on the oracles $H_0, \ldots, H_k$, and taking the adjacent pair $(k^* - 1, k^*)$ for which there were the most differences in output.

Note that the teal color in the protocol diagram indicates (updating of) the prover's state.

Lemma 4.3. *Protocol 2 is perfectly complete.*

Proof. This is clear since $\mathsf{lookup}^G_{n,n}$ is deterministic. □

We first prove that Protocol 2 is actively secure, which serves as a stepping-stone to the proof of sequential MIM security.

Lemma 4.4. *If G is a PRG, Protocol 2 is secure against active attacks.*

Proof. In the following, let e_j denote the noise string for index j. We begin by establishing that e_j are indistinguishable from uniformly random noise. Consider the following games:

Game 1. $\mathcal{P}, \mathcal{V}$ and the adversary $\mathcal{A}$ play the active security game.

Game 2. $\mathcal{P}, \mathcal{V}$ and the adversary $\mathcal{A}$ play the active security game as before, except that $\mathcal{P}, \mathcal{V}$ no longer have access to the key s': instead, they have oracle access to $\mathsf{lookup}^G_{n,n}(s', \cdot)$.

Game 3. Like Game 2, except that the oracle $\mathsf{lookup}^G_{n,n}(s', \cdot)$ is replaced by a random oracle.

It is clear that Games 1 and 2 are perfectly indistinguishable from the adversary's point of view, since the messages sent by $\mathcal{P}, \mathcal{V}$ are distributed identically in the two games. Now suppose, for contradiction, that there exists an adversary $\mathcal{A}$ which can efficiently distinguish between Games 2 and 3. Then, this adversary could be used to efficiently distinguish between (oracle access to) $\mathsf{lookup}^G_{n,n}(s', \cdot)$ and a random oracle – this contradicts Theorem 4.2. Therefore, Games 1, 2, and 3 are computationally indistinguishable from each other.

We have established that the prover's message $z = a \cdot s + e_j$ is indistinguishable from $a \cdot s + r$ for random r, Hence, z is indistinguishable from random to any active adversary (even after observing polynomially many honest transcripts), regardless of the adversary's choice of a. So, to prove active security, it remains only to consider the interaction of $\mathcal{A}$ with the verifier. Given a challenge a from an honest verifier $\mathcal{V}$, we argue that $\mathcal{A}$ can have no more than negligible advantage at guessing the (unique) value of z that will cause $\mathcal{V}$ to accept, by considering the following two cases:

1. $\mathcal{A}$ sends an index i that was not used when talking to the honest prover. In this case, we could give the adversary the e values for this i for free (as it is independent of the what happens for the other indices). Now the adversary's task is equivalent to guessing $a \cdot s$, which he cannot do since he has no information about s.

2. $\mathcal{A}$ sends an index i that was previously used in a query to the prover. Let z, i be the response (to a) from the honest prover. Say the honest verifier sends a' and let z', i be the adversary's response. If there is a non-negligible probability that z' is accepted, then it follows that $z - z' = (a - a') \cdot s$, and thus $s = (z - z')(a - a')^{-1}$. This happens with negligible probability since all of a, a', z, z' were chosen independently of s. □

Theorem 4.5. *If G is a PRG, Protocol 2 is secure against sequential man-in-the-middle attacks.*

Proof. We show that given an adversary $\mathcal{A}$ which achieves a certain advantage when conducting a sequential MIM attack, it is possible to build a new adversary $\mathcal{A}'$ that *only talks to the prover* in Protocol 2 and achieves essentially the same advantage. First, we will replace the honest verifier by a fake verifier $\mathcal{V}'$ who has no access to s or the e_j but still gives essentially the same answers as $\mathcal{V}$. Then we argue that for any sequential man-in-the-middle attack, there is an equally successful *active* attack, and finally refer to Lemma 4.4 for the active security of the protocol.

$\mathcal{V}'$ works as follows: when queried by $\mathcal{A}$, it chooses a random challenge a (just like $\mathcal{V}$ does). When $\mathcal{A}$ returns an answer z, j (i.e. the second protocol message), there are two cases to consider:

1. $\mathcal{A}$ previously received answer z', j from $\mathcal{P}$, where $z' = a' \cdot s + e_j$ and a' is $\mathcal{A}$'s query to $\mathcal{P}$. Here we have two possibilities:

(a) $a = a'$ (which could be the case if $\mathcal{A}$ queried $\mathcal{P}$ during the current protocol execution): in this case, if $z = z'$, $\mathcal{V}'$ accepts; else, it rejects.

(b) $a \neq a'$: $\mathcal{V}'$ always rejects.

2. $\mathcal{A}$ never previously received an answer of the form z', j from $\mathcal{P}$. In this case $\mathcal{V}'$ always rejects.

Consider the first time $\mathcal{A}$ queries the verifier. The challenge produced by $\mathcal{V}$ has exactly the same distribution as the one $\mathcal{V}'$ outputs. Now, in case 1a, notice that $\mathcal{V}$ will accept if and only if z has the correct value $a \cdot s + e_j$: so $\mathcal{V}'$ always makes the same decision as $\mathcal{V}$. In case 1b, $\mathcal{V}$ rejects except with negligible probability, so $\mathcal{V}'$ is statistically close to the right behavior. This is because accepting would imply that $z = a \cdot s + e_j$, but we also have $z' = a' \cdot s + e_j$ so then $s = (z - z')(a - a')^{-1}$. This happens with negligible probability because s is random and z, z', a, a' are all independent of s. This holds because all of $\mathcal{P}$'s responses (including z') are independent of s. Moreover, since this is the first query, $\mathcal{V}$ has not even looked at s yet, so a, a' and z must be independent of s too. Finally, in case 2, note that no one sees e_j before the adversary produces z, j. If $\mathcal{V}$ accepts, we have $z = a \cdot s + e_j$, so $e_j = z - a \cdot s$, which happens with negligible probability since z, a and s are independent of e_j.

Therefore, we can modify $\mathcal{V}$ so that it behaves like $\mathcal{V}'$ for the first query, and the adversary's advantage changes at most negligibly as a result. Repeating the same argument for all the queries, we reach the game where $\mathcal{V}$ is entirely replaced by $\mathcal{V}'$, and the adversary's advantage is still at most negligibly different from in the original game.

Since $\mathcal{V}'$ does not possess any secret information, an adversary can run $\mathcal{V}'$ "in his head". So for any adversary $\mathcal{A}$ which has non-negligible advantage in a man-in-the-middle attack, we can construct an adversary $\mathcal{A}'$ that emulates both $\mathcal{A}$ and $\mathcal{V}'$ "in his head" and achieves the same advantage, but conducting an *active* attack (since he need not interact with the real verifier $\mathcal{V}$).

Finally, the security of the protocol against sequential man-in-the-middle attacks follows from the security against active attacks, which was shown in Lemma 4.4. □

Building upon the ideas of Protocol 2, our next protocol achieves ℓ-concurrent MIM security for any polynomial ℓ, at very little extra cost.

4.2.2 ℓ-concurrent man-in-the-middle secure protocol

In the concurrent MIM secure version of our protocol, we use ℓ-independent hashing, defined below.

Public parameters. PRG G, security parameter $n \in \mathbb{Z}$, concurrency parameter $\ell = \mathsf{poly}(n)$, function family $\mathcal{H} = \{h_r\}_{r \in \{0,1\}^\beta}$ of 2ℓ-wise independent hash functions mapping $(\ell+1) \cdot n$ bits to n bits.

Key generation. $\mathsf{Gen}(1^n)$ samples $s \leftarrow R, s' \leftarrow \{0,1\}^n$ and outputs secret key (s, s').

Initial state. The prover's state consists of $i \in \mathbb{N}$ initialised to 1.

$\mathcal{P}(s, s'; i)$		$\mathcal{V}(s, s')$
	$\xleftarrow{\quad a \quad}$	$a \leftarrow R$
$r_i := \mathsf{lookup}^G_{n,(\ell+1)\cdot n+\beta}(s', i)$		$r_i := \mathsf{lookup}^G_{n,(\ell+1)\cdot n+\beta}(s', i)$
$e := h_{r_i^{[1,\beta]}}\left(a + r_i^{[\beta+1,(\ell+1)\cdot n+\beta]}\right)$		accept iff $z + a \cdot s =$
$z := a \cdot s + e$	$\xrightarrow{\quad z,\ i \quad}$	$h_{r_i^{[1,\beta]}}\left(a + r_i^{[\beta+1,(\ell+1)\cdot n+\beta]}\right)$
$i := i + 1$		

Protocol 3: ℓ-concurrent MIM secure protocol based on any PRG

Definition 4.6 (ℓ-independent hash function family)**.** *Let $\mathcal{H}$ be a family of hash functions, where each hash function $h \in \mathcal{H}$ maps from a domain $\{0,1\}^n$ to a codomain $\{0,1\}^m$. $\mathcal{H}$ is said to be* ℓ-independent *if for any fixed sequence of ℓ distinct inputs $(k_1, \ldots, k_\ell) \in D^\ell$ and for $h \leftarrow \mathcal{H}$ chosen at random, the sequence $(h(x_1), \ldots, h(x_\ell))$ is uniformly random in C^ℓ.*

Note that a hash function as required by the protocol can be chosen as a random polynomial of degree at most 2ℓ over the field with 2^n elements. To compute the hash function, evaluate the polynomial at the input point, and output the least significant m bits of the result. In this paper, we will assume that this is how all our hash functions are instantiated.

Lemma 4.7. *Protocol 3 is perfectly complete.*

Proof. This is clear since $\mathsf{lookup}^G_{n,(\ell+1)\cdot n+\beta}$ is deterministic. □

Our security proofs follow a similar structure to those of Protocol 2: we first prove (concurrent) active security, then use this to prove man-in-the-middle security. Additionally, we require the following technical lemma, which can be seen as a generalization of the leftover hash lemma and has a similar proof.

Lemma 4.8 ([DFMV13])**.** *Let $(X_1, X_2, \ldots, X_\ell) \in \mathcal{X}^\ell$ be ℓ (possibly dependent) random variables such that $H_\infty(X_i) \geq \gamma$ and $(X_1, \ldots, X_\ell)$ are pairwise different. Let $\mathcal{H} = \{h : \mathcal{X} \to \mathcal{Y}\}$ be a family of 2ℓ-wise independent hash functions, with $|\mathcal{Y}| = 2^k$. Then for random $h \leftarrow \mathcal{H}$ we have that the statistical distance satisfies*

$$\Delta((h, h(X_1), h(X_2), \ldots, h(X_\ell)); (h, U^1_{\mathcal{Y}}, \ldots, U^\ell_{\mathcal{Y}})) \leq \frac{\ell}{2} \cdot 2^{(\ell \cdot k - \gamma)/2},$$

where $U^1_{\mathcal{Y}}, \ldots, U^\ell_{\mathcal{Y}}$ are ℓ independent and uniformly distributed variables.

Lemma 4.9. *If G is a PRG, Protocol 3 is secure against ℓ-concurrent active attacks.*

Proof. Recall that an ℓ-concurrent active adversary may have concurrent access to up to ℓ honest provers, but as usual, he cannot reset the provers. The updating of the prover's state in Protocol 3 ensures that for any given prover, the value r_i is freshly pseudorandomly sampled for each execution (that is, each value of the counter i). In particular, the hash function seed $r_i^{[1,\beta]}$ is freshly pseudorandomly sampled for each value of i, and this means that for any polynomial number of executions, with overwhelming probably all the hash function seeds will be distinct.

Therefore, an ℓ-concurrent active adversary can obtain at most ℓ outputs of the hash function h_s for any given seed s. For any $i \in \mathbb{N}$, let $s \stackrel{\text{def}}{=} r_i^{[1,\beta]}$ be the corresponding seed, and let $x \stackrel{\text{def}}{=} r_i^{[\beta+1,(\ell+1)\cdot n+\beta]}$ be the summand inside the hash function argument. Suppose that the adversary obtains samples $h_s(a_1 + x), \ldots, h_s(a_\ell + x)$ from the honest provers. If the adversary chooses some a_i, a_j to be equal, then the samples $h_s(a_i + x), h_s(a_j + x)$ will also be equal, so the adversary will not gain more information than if he made just one query a_i. Hence, we assume without loss of generality that the $a_1, \ldots, a_\ell$ are distinct.

Since G is a PRG, we can replace s and x with uniformly randomly chosen $s' \leftarrow \{0,1\}^\beta$ and $x' \leftarrow \{0,1\}^{(\ell+1)\cdot n}$, and the adversary's advantage will change at most negligibly. Let $U_{(\ell+1)\cdot n}$ be a random variable that is uniformly distributed over the set of $((\ell+1)\cdot n)$-bit strings. Consider the random variables $U_{(\ell+1)\cdot n} + a_1, U_{(\ell+1)\cdot n} + a_2, \ldots, U_{(\ell+1)\cdot n} + a_\ell$. Each variable $U_{(\ell+1)\cdot n} + a_i$ has entropy $H(U_{(\ell+1)\cdot n} + a_i) = H_\infty(U_{(\ell+1)\cdot n} + a_i) = (\ell+1)\cdot n$. They are not independent, but they are pairwise different because the a_i's are distinct. Therefore, by Lemma 4.8,

$$\{h_s, h_s(U_{(\ell+1)\cdot n} + a_1), \ldots, h_S(U_{(\ell+1)\cdot n} + a_\ell)\} \stackrel{s}{\approx} \{h_s, U^1_n, \ldots U^\ell_n\}.$$

(In the notation of the lemma: we have $k = n$ and $\gamma = (\ell+1)\cdot n$, so the distance is $\frac{\ell}{2}2^{-n}$, which is negligible given that $\ell = \mathsf{poly}(n)$.)

Finally, since the seeds $s_i \stackrel{\text{def}}{=} r_i^{[1,\beta]}$ are freshly pseudorandomly sampled for each value of i, the hash functions $h_{s_1}, h_{s_2}, \ldots$ used in the protocol are indistinguishable from independent random hash functions. Therefore, the output of any hash function h_{s_i} is indistinguishable from random even given the outputs of the other hash functions $h_{s_{i'}}, h_{s_{i''}}, \ldots$ from different executions of the protocol. □

Theorem 4.10. *If G is a PRG, Protocol 3 is secure against ℓ-concurrent MIM attacks.*

The proof of Theorem 4.10 is very similar to that of Theorem 4.5, and is given in Appendix B.

Depth of computation. Since the multiplication and hash function sampling and evaluation can be done in depth $O(\log(n))$, it holds that if the PRG G is of poly-logarithmic depth, then Protocols 2 and 3 require only poly-log depth computation.

5 PRG-based authentication via "public randomizers"

In this section we take a different approach to constructing authentication protocols from PRGs, which yields man-in-the-middle secure constructions that are incomparable to the protocols of Section 4. The differences will be discussed in detail later in this section.

Our primary building block is a PRG of a special format, which we call a "PRG with public randomizer". The definition follows.

Definition 5.1 (Pseudorandom generator with public randomizer)**.** *Let $g : \{0,1\}^{n'} \times \{0,1\}^n \to \{0,1\}^{m(n,n')}$ be a deterministic polynomial-time algorithm. g is a* pseudorandom generator with public randomizer *if $m(n,n') \geq n+n'$ and for any efficient distinguisher D that outputs a single bit, it holds that $|\Pr[\mathsf{D}(r',r)=1] - \Pr[\mathsf{D}(a,g(a,s))=1]| \leq \mathsf{negl}(n,n')$, where $r \leftarrow \{0,1\}^{m(n,n')}, r' \leftarrow \{0,1\}^{n'}, s \leftarrow \{0,1\}^n$ are chosen uniformly at random, and the probabilities are taken over r, r', s, and the random coins of D.*

A PRG with public randomizer g can trivially be built from a PRG G by setting $g(a,x) = G(x)$. In the initial exposition of the protocol, we assume a black-box PRG with public randomizer; however, in Section 5.3, we will discuss efficient PRGs with public randomizers from specific assumptions (namely, LPN and subset sum) which are particularly suitable for use in our protocol.

The new protocol makes use of ℓ-independent hashing (as introduced in Section 4). In this case, the hashing is used to build from any PRG (with public randomizer) a (bounded) PRF, which is invoked in our authentication protocols.

5.1 The PRF construction

We start by constructing a WPRF family $\mathcal{F}^{(g,\ell,\mathcal{H})}$, parametrized by: a PRG with public randomizer $g : \{0,1\}^{n'} \times \{0,1\}^n \to \{0,1\}^m$, a concurrency parameter $\ell = \mathsf{poly}(n)$, and a family $\mathcal{H} = \{h_r\}_{r \in \{0,1\}^\beta}$ of 2ℓ-wise independent hash functions mapping $(\ell+1)\cdot n$ bits to n bits. We require that $m \geq (\ell+1)\cdot n + \beta$. The input to the WPRF will be of the form $(a,s) \leftarrow \{0,1\}^{n'} \times \{0,1\}^n$.

Now we define $\mathcal{F}^{(g,\ell,\mathcal{H})} = \{F_s\}$ as follows:

$$F_s(a) = g\left(a, h_{s'}(x+a)\right),$$

where $x = g(a,s)^{[1,(\ell+1)\cdot n]} \in \{0,1\}^{(\ell+1)\cdot n}$ and $s' = g(a,s)^{[(\ell+1)\cdot n+1,(\ell+1)\cdot n+1+\beta]} \in \{0,1\}^\beta$.

The next theorem proves that our construction is a WPRF if the adversary makes at most ℓ queries, and after this we consider what happens for more than ℓ queries[10].

[10] Note that if we were *only* interested in security for ℓ queries this could be done much more simply: we could use an ℓ-wise independent hash function family $\mathcal{H} = \{h_s\}_s$, set some s^* as our secret key and output $h_{s^*}(x)$ on input x. However, such a construction would be completely insecure if more than ℓ queries are asked.

Theorem 5.2. *If G is a PRG, $\mathcal{F}^{(g,\ell,\mathcal{H})}$ is a weak pseudorandom function family for any adversary making no more than ℓ queries to its oracle.*

Proof. Since g is a PRG with public randomizer, we can modify the oracle $\mathcal{O}^{F_K}$ from Definition 2.8 so that it uses uniformly chosen x and h (rather than choosing them according to the output of g), and the advantage of any adversary will change by at most a negligible amount.

Let $a_1, \ldots, a_\ell$ be the (uniformly random) inputs to the oracle. Let $U_{(\ell+1)\cdot n}$ be a random variable that is uniformly distributed over the set of $((\ell+1)\cdot n)$-bit strings. Consider the random variables $U_{(\ell+1)\cdot n} + a_1, U_{(\ell+1)\cdot n} + a_2, \ldots, U_{(\ell+1)\cdot n} + a_\ell$. Each variable $U_{(\ell+1)\cdot n} + a_i$ has entropy $H(U_{(\ell+1)\cdot n} + a_i) = H_\infty(U_{(\ell+1)\cdot n} + a_i) = (\ell+1)\cdot n$. They are not independent, but they are pairwise different because the a_i's are distinct (with overwhelming probability). Therefore, by Lemma 4.8, $\{h_s, h_s(U_{(\ell+1)\cdot n} + a_1), \ldots, h_S(U_{(\ell+1)\cdot n} + a_\ell)\} \stackrel{s}{\approx} \{h_s, U_m^1, \ldots U_n^\ell\}$. (In the notation of the lemma: we have $k = n$ and $\gamma = (\ell+1)\cdot n$, so the distance is $\frac{\ell}{2}2^{-n}$, which is negligible given that $\ell = \mathsf{poly}(n)$.) We can therefore modify oracle $\mathcal{O}^{F_K}$ again so it outputs $(a_i, g(a_i, e_i))$ where e_i is chosen as a fresh random value for each query. Finally, we can replace the second component by a uniformly random value since g is a PRG with public randomizer. Therefore, the final modified $\mathcal{O}^{F_K}$ is computationally indistinguishable from $\mathcal{O}^{\mathcal{R}}$ where $\mathcal{R}$ is a random oracle. □

Note that the choice of ℓ does not affect the required secret key size for the WPRF.

Our next step is to construct a PRF from this WPRF, for which we require the following result of [NR99]. Their result refers to the concept of a *synthesizer*, which was introduced in their paper (along with weak PRFs). For our construction, it is not necessary to formally define a synthesizer, so we refer the interested reader to [NR99] for further details.

Theorem 5.3 ([NR99])**.** *A secure PRF can be constructed from any synthesizer, and a synthesizer can be constructed from any WPRF whose input and output sets are equal. Furthermore, if the WPRF can be evaluated in poly-logarithmic depth, then so can the resulting PRF.*

It follows from the construction of [NR99] that if we build a PRF from a WPRF using their method, and the adversary makes no more than ℓ queries to the PRF, then no more than ℓ queries are made to any instance of the WPRF (their construction uses several instances of the WPRF with different keys). Therefore, since $\mathcal{F}^{(g,\ell,\mathcal{H})}$ is a secure ℓ-bounded WPRF, the function family $\widetilde{\mathcal{F}}^{(g,\ell,\mathcal{H})}$ obtained by applying Theorem 5.3 to $\mathcal{F}^{(g,\ell,\mathcal{H})}$ is a secure ℓ-bounded PRF.

Such ℓ-bounded (weak) PRFs can already be useful in certain settings, where there is a guaranteed maximum number of queries that will be made to the PRF during its use. Our authentication protocol in Section 5.2 is an example of such an application.

In some settings, however, such a bound on the number of queries is not a realistic assumption. However, all is not lost: we can actually prove security of the WPRF $\mathcal{F}^{(\},\ell,\mathcal{H})}$ in general, if the PRG g has a stronger property: namely, that even if the second elements (the "keys") s of the input to $g(a, s)$ are only ℓ-wise independent between samples, then as long as ℓ is large enough, it is still hard to distinguish any polynomial number of samples from uniformly random. In this case, by Theorem 5.3, we obtain that $\widetilde{\mathcal{F}}^{(g,\ell,\mathcal{H})}$ is a secure PRF. Proofs of (weak) PRF security for $\mathcal{F}^{(g,\ell,\mathcal{H})}$ and $\widetilde{\mathcal{F}}^{(g,\ell,\mathcal{H})}$ when there is no bound on the number of queries are given in Appendix C.

We now give a formal definition of the additional property required of g.

Definition 5.4. *Let $\mathcal{D}_\ell^n(a_1, \ldots, a_k)$ be the distribution defined by the following sampling procedure: choose a random $u \leftarrow \mathbb{Z}_2^{(\ell+1)\cdot n}$ and a 2ℓ-wise independent hash function h mapping $(\ell+1)\cdot n$ bits to n bits (a random polynomial, as described above), and then output $(h(u + a_1), \ldots, h(u + a_k))$.*

Property 5.5. *Let κ be a security parameter for a PRG with randomizer $g : \{0,1\}^{n'} \times \{0,1\}^n \to \{0,1\}^m$. For any $k = \mathsf{poly}(\kappa)$, let $a_1, \ldots, a_k \leftarrow \{0,1\}^{n'}$ and $(s_1, \ldots, s_k) \leftarrow \mathcal{D}_\ell^n(a_1, \ldots, a_k)$. Then there exists a polynomial f such that if we set $\ell = f(\kappa)$, then it holds that*

$$((a_1, g(a_1, s_1)), \ldots, (a_k, g(a_k, s_k))) \stackrel{c}{\approx} ((a_1, u_1), \ldots, (a_k, u_k))$$

where the $u_i \leftarrow \{0,1\}^m$ are uniformly random.

We show the security of the PRF $\widetilde{\mathcal{F}}^{(g,\ell,\mathcal{H})}$ in general (that is, for any polynomial number of queries). The proofs of Theorems 5.6 and 5.7 are given in Appendix C.

Theorem 5.6. *If g is a PRG with public randomizer that satisfies Property 5.5, then $\mathcal{F}^{(g,\ell,\mathcal{H})}$ is a secure WPRF.*

Theorem 5.7. *Let $\widetilde{\mathcal{F}}^{(g,\ell,\mathcal{H})} = \{\widetilde{F}_s\}$ be the function family obtained by applying the [NR99] technique to the function family $\mathcal{F}^{(g,\ell,\mathcal{H})}$ defined above, for parameters $n, \ell \in \mathbb{Z}$ and $\mathcal{H}$ a family of 2ℓ-wise independent hash functions mapping $(\ell+1)\cdot n$ bits to n bits. Then, if g is a PRG with public randomizer, $\widetilde{\mathcal{F}}^{(g,\ell,\mathcal{H})}$ is a secure PRF family for any adversary making at most ℓ queries. If g furthermore satisfies Property 5.5, then $\widetilde{\mathcal{F}}^{(g,\ell,\mathcal{H})}$ is a secure PRF family (for any* PPT *adversary).*

It is not known whether (some) existing PRG constructions satisfy Property 5.5 (under their respective hardness assumptions). While we cannot give a concrete function that provably has the property, we give a couple of arguments in favor of the plausibility of PRGs that satisfy the property. For any polynomial value of k, the probability that the a_i's are not distinct is negligible, and therefore by Lemma 4.8 the e_i's are ℓ-wise independent. Hence, an attack must consider at least $\ell+1$ of the k instances simultaneously. Furthermore, the ℓ-wise independence holds even if the hash function h_s is given. In our setting the adversary does not get h_s, so this should add to the difficulty of the problem.

Even in the absence of a definite candidate function, characterizing this property allows for the "graceful degradation" of our security guarantee for Protocol 4 in the case that more than ℓ queries are made by the adversary: the enhanced guarantee is that if the PRG g used to build the function family $\mathcal{F}$ additionally satisfies Property 5.5, then $\mathcal{F}$ is a WPRF secure against *any* polynomial-time adversary, and thus $\widetilde{\mathcal{F}}$ is a PRF secure against any efficient adversary too. When using PRGs based on specific hardness assumptions, having the property translates to a stronger ℓ-wise independence assumption on the hardness of the underlying problem, as we see in Section 5.3.

5.2 The protocol

We now present a new authentication protocol based on the PRF construction from Section 5.1. This can be based on any PRG with public randomizer, which in turn can be based on any PRG. Let G be the PRG underlying our construction, and let g be a PRG with public randomizer (such as the simple example $g : (a, s) \mapsto G(s)$ which is based on G).

Public parameters. PRG G, security parameter $n \in \mathbb{Z}$, concurrency parameters $\ell, \ell' = \mathsf{poly}(n)$ with $\ell' < \ell$, function family $\mathcal{H} = \{h_r\}_{r\in\{0,1\}^\beta}$ of 2ℓ-wise independent hash functions mapping $(\ell+1)\cdot n$ bits to n bits.

Key generation. $\mathsf{Gen}(1^\kappa)$ samples secret key $s_0 \leftarrow R$.

Initial state. The prover's state consists of $i, j \in \mathbb{Z}$, both initialized to 1.

$\mathcal{P}(s_0; i, j)$		$\mathcal{V}(s_0)$
	$\xleftarrow{a}$	$a \leftarrow R$
$s := \mathsf{lookup}^G_{n,n}(s_0, i)$		$s := \mathsf{lookup}^G_{n,n}(s_0, i)$
$z := \widetilde{F}_s(a)$	$\xrightarrow{z,\ i}$	accept iff $z = \widetilde{F}_s(a)$
$j := j + 1$		
if $j = \ell' + 1$ then $\{i := i + 1; j = 1\}$		

Protocol 4: MIM secure protocol based on (bounded) PRF from any PRG

In the below protocol, $\widetilde{F}_s$ is the function belonging to the PRF $\widetilde{\mathcal{F}}^{(g,\ell,\mathcal{H})}$, with key s.

Lemma 5.8. *Protocol 4 is perfectly complete.*

Proof. This is clear since $\mathsf{lookup}_{n,n}^{G}$ and $\widetilde{\mathcal{F}}_s$ are deterministic. □

Theorem 5.9. *If g is a PRG with public randomizer, Protocol 4 is secure against $\lfloor \ell/\ell' \rfloor$-concurrent man-in-the-middle attacks.*

Proof. G is a PRG, so by Lemma 4.2, since the secret s_0 is chosen at random, the values $s = \mathsf{lookup}_{G_n}(s_0, i)$ for $i = 1, 2, \ldots$ used as PRF seeds in the protocol are indistinguishable from random. By Theorem 5.7, $\widetilde{\mathcal{F}}^{(g,\ell,\mathcal{H})}$ is a secure ℓ-bounded PRF family. The updating of the prover's state in Protocol 4 guarantees that for *each* prover that he interacts with, the adversary makes no more than $\ell' < \ell$ queries to $\widetilde{F}_s$ for any given seed s. Hence, interacting (concurrently) with up to $\lfloor \ell/\ell' \rfloor$ copies of the honest prover will give the adversary at most ℓ queries to $\widetilde{F}_s$ for any given seed s, and so the prover responses z are indistinguishable from outputs of a random oracle. Therefore, the adversary can gain no more than negligible advantage from interacting with the provers.

The other option available to the adversary is to interact with up to $\lfloor \ell/\ell' \rfloor$ honest verifiers $\mathcal{V}$ and observe their accept/reject decisions. However, the adversary gains no advantage from interacting with more than one verifier, since the verifier's accept/reject decision is completely determined by the protocol transcript and the secret key. (For a formal proof of the last statement, see Lemma B.1 in Appendix B.) Now we consider whether the adversary can gain an advantage from access to the accept/reject decisions of a single honest verifier. For initial message a, the verifier $\mathcal{V}$ accepts if and only if $z = \widetilde{F}_s(a)$. $\mathcal{V}$'s response for any given input a is distributed as one of these two cases:

1. If $\mathcal{A}$ already queried the prover $\mathcal{P}$ on input a, then $\mathcal{A}$ already knows (given a) the true distribution of z, so the verifier's response gives the adversary no additional information.

2. Otherwise, if $\mathcal{A}$ has not queried $\mathcal{P}$ on input a, then since $\mathcal{F}^{(g,\ell,\mathcal{H})}$ is a PRF family (for up to ℓ queries) and s is randomly chosen and unknown to $\mathcal{A}$, the verifier's decision on any transcript (a, z') where z' is chosen by $\mathcal{A}$ is indistinguishable, to the adversary, from the decision procedure that always outputs reject. So, again, the verifier's response gives the adversary no additional information.

Let D_1 denote the distribution defined in item 1, and D_2 denote the distribution defined in item 2 above. Now consider an adversary who makes polynomially many queries $a_1, \ldots, a_k$. In this case, by a hybrid argument replacing each a_i by b_i for each i one by one, we argue that the joint distribution of the verifier's responses to all the queries is indistinguishable from the joint distribution of $b_1, \ldots, b_k$ where each b_i is drawn (independently) from either D_1 or D_2. It follows that the adversary who makes polynomially many queries gains at most negligible advantage from interacting with $\mathcal{V}$. Therefore, the protocol is secure against $\lfloor \ell/\ell' \rfloor$-concurrent MIM attacks. □

Theorem 5.10. *If g is a PRG with public randomizer that satisfies Property 5.5, Protocol 4 is secure against fully concurrent man-in-the-middle attacks, even for adversaries with the power to reset the prover during the query stage.*

Proof. By Theorem 5.7, since g is a PRG with public randomizer and satisfies Property 5.5, $\widetilde{\mathcal{F}}^{(g,\ell,\mathcal{H})}$ is a secure PRF family. The rest of the proof is exactly as in the proof of Theorem 5.9. □

5.3 Non-black-box constructions from LPN and subset sum

Having designed Protocol 4 in a black-box way with the aim of generality and efficiency, a natural next step is to consider what concrete low-depth PRGs it might be instantiated with.

An LPN-based solution would be a first candidate, since LPN has already been well-studied as a building block for lightweight authentication, and efficient PRGs based on LPN are known.

5.3.1 Public randomizers based on LPN

The original LPN-based pseudorandom generator of [BFKL94] takes a (uniformly random) input r and uses it to sample A, s uniformly, and e with each bit distributed as Ber_τ, and then outputs $(A, As+e)$. Building upon this, [ACPS09] constructed an efficient (quasi-linear time) linear-stretch PRG based on the LPN problem. In particular, their construction makes use of the fact that $s \in \mathbb{Z}_2^l$ and $e \in \mathbb{Z}_2^m$ together have $l + m \cdot H(\tau)$ bits of entropy, and can be very efficiently sampled using roughly that many random bits by a method of [AIK08]. (A can be fixed as a public parameter and thus need not be sampled afresh each time.) However, the [ACPS09] construction is defined only for $\tau = 2^{-t}$ for $t \in \mathbb{N}$. In Appendix D, we give a generalization of their construction for any real-valued $\tau \in (0, \frac{1}{2}]$. Let $G^{\mathsf{LPN}}_{n,\tau}$ denote the LPN-based PRG described in Appendix D, for parameters $n \in \mathbb{N}, \tau \in (0, \frac{1}{2}]$. Note that $G^{\mathsf{LPN}}_{n,\tau}$ takes as input a random triple (A, s, e) and outputs $(A, As + e')$ where e' is obtained by sampling Bernoulli-distributed bits using e.

While Protocol 4 could, of course, be built using this PRG $G^{\mathsf{LPN}}_{n,\tau}$ as a black box, we can in fact do better by making use of the PRG's internal structure. In the black-box case, the protocol would be ℓ-concurrent MIM secure assuming that the LPN problem is hard; and Property 5.5 would correspond to the assumption that the LPN problem is hard even when the triple (A, s, e) is sampled using a 2ℓ-wise independent hash function rather than uniformly at random.

However, since $G^{\mathsf{LPN}}_{n,\tau}$ is already a PRG with public randomizer, the function g in the construction of Protocol 4 can be instantiated directly by $G^{\mathsf{LPN}}_{n,\tau}$. Then, as before, the resulting protocol is ℓ-concurrent MIM secure assuming that the LPN problem is hard. Moreover, full concurrent MIM security can be achieved with a weaker assumption this time, which does not involve A. Formally, the assumption is that the following conjecture holds – this may be considered a "stronger variant" of the standard LPN assumption.

Definition 5.11. *Let $\mathcal{D}^{n,\tau}_\ell(a_1, \ldots, a_k)$ be the distribution defined by the following sampling procedure: choose a random $u \leftarrow \mathbb{Z}_2^{(\ell+1)\cdot n}$ and a 2ℓ-wise independent hash function h with input size $(\ell + 1) \cdot n$ bits and output size n bits (a polynomial, as described above), and then output*

$$(\mathsf{BerSamp}^n_\tau(h(u + a_1)), \ldots, \mathsf{BerSamp}^n_\tau(h(u + a_k)))$$

(with zero-padding where necessary to match summand lengths).

In the above, $\mathsf{BerSamp}^n_\tau$ is a function for sampling from Ber^n_τ which takes as input a short (pseudo)random vector. For a formal definition of the sampling function, refer to Appendix D.

Conjecture 5.12. *Let κ be a security parameter, let $n, k = \mathsf{poly}(\kappa)$ and $\tau \in (0, \frac{1}{2}]$. Let $a_1, \ldots, a_k \leftarrow \{0,1\}^m \times n$ and $s \leftarrow \{0,1\}^n$, and $e_1, \ldots, e_k \leftarrow \mathcal{D}^{n,\tau}_\ell(a_1, \ldots, a_k)$. Then there exists a polynomial f such that if we set $\ell = f(\kappa)$, then it holds that*

$$((a_1, a_1 \cdot s + e_1), \ldots, (a_k, a_k \cdot s + e_k)) \overset{c}{\approx} ((a_1, u_1), \ldots, (a_k, u_k))$$

where the $u_i \leftarrow \{0,1\}^m$ are uniformly random.

Of course, this assumption is new and should be studied carefully before it is used. As an argument in favor of the assumption, note that the a_i's will be distinct with overwhelming probability, and therefore (by Lemma 4.8) the e_i's are ℓ-wise independent. Hence, any attack that is faster than trying to solve LPN directly must consider at least $\ell + 1$ of the k instances simultaneously. Moreover, as noted in Section 5.1, the ℓ-wise independence holds even if the hash function h is known – but the adversary in our protocol does not even know h.

In the context of LPN-based authentication. In existing LPN-based schemes, though any constant noise rate $\tau \in (0, \frac{1}{2})$ is supported, the completeness guarantees are asymptotic: in fact, as $\tau \to \frac{1}{2}$, the tradeoff between efficiency and completeness is rather poor. This is because these

schemes all involve a "threshold check" similar to the one in Protocol 1. Concretely, for $\tau = \frac{1}{2} - \varepsilon$, if we want a completeness error of at most e^{-w} for some fixed w, then by a Chernoff bound, $n = O(1/\varepsilon^2)$. Hence, to get arbitrarily low error probability we must have extremely large n (and thus, very slow algorithms and large memory requirements). This is undesirable since we want τ close to $\frac{1}{2}$ in order to use the weakest hardness assumption possible[11].

In contrast, by making use of an LPN-based PRG as in our protocols, we diverge from the "threshold check" paradigm which was introduced by [HB01] and has remained pervasive in the LPN-based authentication literature. Our protocols are perfectly complete, and moreover, as $\tau \to \frac{1}{2}$, the communication complexity of our protocols remains unaffected (although computational complexity of the PRG increases).

In the context of LPN-based PRFs. LPN is particularly appropriate for consideration in our setting because it has the interesting properties that it is a promising assumption from a practical perspective, and efficient LPN-based PRGs are known [BFKL94; ACPS09], yet it is not known whether (even weak) PRFs can be constructed from LPN with better efficiency than the rather slow [GGM86] construction. This, for example, makes LPN ineligible for effective use with the WPRF-based authentication protocols of [DKPW12; LM13], at least from our current state of knowledge. In this light, we remark that the ℓ-bounded PRF of Section 5.1 could be of broader interest to serve as an LPN-based PRF also in other applications, where it is reasonable to place a bound on the number of queries.

5.3.2 Construction based on subset sum

The subset sum problem is in a somewhat similar situation to LPN, in that while an efficient SS-based PRG is known, the best known PRF construction is again via [GGM86] and thus too slow to be useful in many applications. We find that the SS-based PRG of [IN96] is a PRG with public randomizer that can be used directly to instantiate the function g in our protocol construction, allowing security under a milder assumption than by the black-box method. Details of the SS-based PRG with public randomizer may be found in Appendix E. To our knowledge, the subset sum problem has not been proposed for use in the authentication setting before.

References

[App13] Benny Applebaum. "Pseudorandom generators with long stretch and low locality from random local one-way functions". In: *SIAM Journal on Computing* 42.5 (2013), pp. 2008–2037.

[ACPS09] Benny Applebaum, David Cash, Chris Peikert, and Amit Sahai. "Fast Cryptographic Primitives and Circular-Secure Encryption Based on Hard Learning Problems". In: *Advances in Cryptology - CRYPTO 2009*. Ed. by Shai Halevi. Vol. 5677. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2009, pp. 595–618. ISBN: 978-3-642-03355-1. DOI: 10.1007/978-3-642-03356-8_35. URL: http://dx.doi.org/10.1007/978-3-642-03356-8_35.

[AIK08] Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. "On Pseudorandom Generators with Linear Stretch in NC^0". In: *Computational Complexity* 17.1 (2008), pp. 38–69.

[BDOZ11] Rikke Bendlin, Ivan Damgård, Claudio Orlandi, and Sarah Zakarias. "Semi-homomorphic Encryption and Multiparty Computation". In: *EUROCRYPT*. Ed. by Kenneth G. Paterson. Vol. 6632. Lecture Notes in Computer Science. Springer, 2011, pp. 169–188. ISBN: 978-3-642-20464-7.

[11] Note that when $\tau = \frac{1}{2}$, LPN samples are information-theoretically indistinguishable from random.

[BFKL94] Avrim Blum, Merrick L. Furst, Michael J. Kearns, and Richard J. Lipton. "Cryptographic Primitives Based on Hard Learning Problems". In: *Proceedings of the 13th Annual International Cryptology Conference on Advances in Cryptology*. CRYPTO '93. London, UK, UK: Springer-Verlag, 1994, pp. 278–291. ISBN: 3-540-57766-1. URL: http://dl.acm.org/citation.cfm?id=646758.759585.

[DFMV13] Ivan Damgård, Sebastian Faust, Pratyay Mukherjee, and Daniele Venturi. "Tamper Resilient Cryptography Without Self-Destruct". In: *IACR Cryptology ePrint Archive* 2013 (2013), p. 124.

[DKPW12] Yevgeniy Dodis, Eike Kiltz, Krzysztof Pietrzak, and Daniel Wichs. "Message Authentication, Revisited". In: *EUROCRYPT*. 2012, pp. 355–374.

[GGM86] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. "How to construct random functions". In: *J. ACM* 33.4 (1986), pp. 792–807.

[Hey+12] Stefan Heyse et al. "Lapin: An Efficient Authentication Protocol Based on Ring-LPN". In: *FSE*. Ed. by Anne Canteaut. Vol. 7549. Lecture Notes in Computer Science. Springer, 2012, pp. 346–365. ISBN: 978-3-642-34046-8.

[HB01] NicholasJ. Hopper and Manuel Blum. "Secure Human Identification Protocols". English. In: *Advances in Cryptology ASIACRYPT 2001*. Ed. by Colin Boyd. Vol. 2248. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2001, pp. 52–66. ISBN: 978-3-540-42987-6. DOI: 10.1007/3-540-45682-1_4. URL: http://dx.doi.org/10.1007/3-540-45682-1_4.

[IN96] Russell Impagliazzo and Moni Naor. "Efficient Cryptographic Schemes Provably as Secure as Subset Sum". In: *J. Cryptology* 9.4 (1996), pp. 199–216.

[JW05] Ari Juels and Stephen A. Weis. "Authenticating Pervasive Devices with Human Protocols". In: *CRYPTO*. Ed. by Victor Shoup. Vol. 3621. Lecture Notes in Computer Science. Springer, 2005, pp. 293–308. ISBN: 3-540-28114-2.

[KSS10] Jonathan Katz, Ji Sun Shin, and Adam Smith. "Parallel and Concurrent Security of the HB and HB$^+$ Protocols". In: *J. Cryptology* 23.3 (2010), pp. 402–421.

[Kil+11] Eike Kiltz et al. "Efficient Authentication from Hard Learning Problems". In: *EUROCRYPT*. Ed. by Kenneth G. Paterson. Vol. 6632. Lecture Notes in Computer Science. Springer, 2011, pp. 7–26. ISBN: 978-3-642-20464-7.

[LM13] Vadim Lyubashevsky and Daniel Masny. "Man-in-the-Middle Secure Authentication Schemes from LPN and Weak PRFs". English. In: *Advances in Cryptology CRYPTO 2013*. Ed. by Ran Canetti and JuanA. Garay. Vol. 8043. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2013, pp. 308–325. ISBN: 978-3-642-40083-4. DOI: 10.1007/978-3-642-40084-1_18. URL: http://dx.doi.org/10.1007/978-3-642-40084-1_18.

[NR99] Moni Naor and Omer Reingold. "Synthesizers and Their Application to the Parallel Construction of Pseudo-Random Functions". In: *J. Comput. Syst. Sci.* 58.2 (1999), pp. 336–375.

[Pat11] Kenneth G. Paterson, ed. *Advances in Cryptology - EUROCRYPT 2011 - 30th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tallinn, Estonia, May 15-19, 2011. Proceedings*. Vol. 6632. Lecture Notes in Computer Science. Springer, 2011. ISBN: 978-3-642-20464-7.

[PP03] David Pointcheval and Guillaume Poupard. "A New NP-Complete Problem and Public-Key Identification". In: *Des. Codes Cryptography* 28.1 (2003), pp. 5–31.

A Formal specification of hybrid $\widetilde{H}_{k^*}$

In this section we give a formal description of the algorithm $\widetilde{H}_{k^*}$ used in the proof of Theorem 4.2. $\widetilde{H}_{k^*}$ takes as input $(\widetilde{r}_1, \ldots, \widetilde{r}_T)$, and then behaves exactly like H_{k^*}, except in the following aspects:

- when $\widetilde{H}_{k^*}$ is initialised, it sets a variable $\mathsf{next} := 0$; and
- if $\lfloor \log_2(i) \rfloor + 1 \geq k^*$ (that is, the depth of the output node for query i is at least k^*) then $\widetilde{H}_{k^*}$ first stores the three tuples

  $$(\mathsf{leaf}, \ell, (\widetilde{r}_{\mathsf{next}})^{[1,m]}), \qquad (\mathsf{root}, \alpha_0, (\widetilde{r}_{\mathsf{next}})^{[m+1,m+n]}), \qquad (\mathsf{root}, \alpha_0 + 1, (\widetilde{r}_{\mathsf{next}})^{[m+n+1,m+2n]}),$$

  where α_0, ℓ are defined by

  $$\alpha_0 = \begin{cases} 2 \cdot (i - 2^{\lfloor \log_2(i) \rfloor} - 1) & \text{if } \lfloor \log_2(i) \rfloor + 1 = k^* \\ 2 \cdot \lfloor \alpha_{(i,k^*)}/2 \rfloor & \text{otherwise} \end{cases},$$

  $$\ell = \begin{cases} i & \text{if } \lfloor \log_2(i) \rfloor + 1 = k^* \\ 2^{(k^*-1)} + (\alpha_0/2) & \text{otherwise} \end{cases};$$

  then $\widetilde{H}_{k^*}$ increments next by 1, and outputs ρ_i, defined by:

  $$\rho_i = \begin{cases} (\widetilde{r}_{\mathsf{next}})^{[1,m]} & \text{if } \lfloor \log_2(i) \rfloor + 1 = k^* \\ \mathsf{lookup}^G_{n,m}\left((\widetilde{r}_{\mathsf{next}})^{[m+1,m+n]}, \gamma_{(i,j^*)}\right) & \text{if } \alpha_{(i,k^*)} = \alpha_0 \\ \mathsf{lookup}^G_{n,m}\left((\widetilde{r}_{\mathsf{next}})^{[m+n+1,m+2n]}, \gamma_{(i,j^*)}\right) & \text{if } \alpha_{(i,k^*)} = \alpha_0 + 1 \end{cases}.$$

In terms of the tree representation of the pseudorandom look-up function: $\widetilde{H}_{k^*}$ behaves exactly like H_{k^*}, except that the values associated with nodes at depth k^* are taken from the input values $\widetilde{r}_1, \ldots, \widetilde{r}_T$. Note that although the number of nodes at depth k^* may be greater than T, next can never become greater than T during an execution of $\widehat{\mathsf{D}}_{\mathsf{PRG}}$, because $\widehat{\mathsf{D}}$ cannot make more than T queries: therefore, $\widetilde{r}_{\mathsf{next}}$ is always well-defined.

B Proof of ℓ-concurrent MIM security of Protocol 3

Lemma B.1. *For any two-round authentication protocol in which the verifier sends the first message, and where the verifier's accept/reject decision is a deterministic function of the secret key, the initial message of the verifier, and the prover's response: it holds that any adversary $\mathcal{A}$ with access to multiple honest verifiers $\mathcal{V}_1, \ldots, \mathcal{V}_\ell$ can be perfectly simulated by another adversary $\mathcal{A}'$ with access to only one honest verifier $\mathcal{V}$*

Proof. The simulation works as follows: $\mathcal{A}'$ runs $\mathcal{A}$, and for every protocol session that $\mathcal{A}$ begins with honest verifier $\mathcal{V}_j$, $\mathcal{A}'$ starts a new session with verifier $\mathcal{V}$ and forwards the initial message a of $\mathcal{V}$ to $\mathcal{A}$. Then, when $\mathcal{A}$ returns to the open session with $\mathcal{V}_j$ and sends a response b, $\mathcal{A}'$ returns to the corresponding session with $\mathcal{V}$ and forwards b to $\mathcal{V}$; and finally, $\mathcal{A}'$ returns to $\mathcal{A}$ the accept/reject decision of $\mathcal{V}$. This is a perfect simulation since for any session, the verifier's decision is a deterministic function of the secret key, the initial message a and the (adversarial) prover's response b. □

Theorem B.2. *If G is a PRG, Protocol 3 is secure against ℓ-concurrent MIM attacks.*

Proof. We show that given an adversary $\mathcal{A}$ which achieves a certain advantage when conducting a concurrent MIM attack, it is possible to build a new adversary $\mathcal{A}''$ that *only talks to the ℓ provers* and achieves essentially the same advantage.

By Lemma B.1, $\mathcal{A}$ can be perfectly simulated with access to just one honest verifier, so we assume henceforth that there is only one honest verifier $\mathcal{V}$. Next, we replace the single honest verifier $\mathcal{V}$ by a fake verifier $\mathcal{V}'$ who has no access to s or the e values, but still gives essentially the same answers as $\mathcal{V}$. Then we argue that for any concurrent man-in-the-middle attack, there is an equally successful concurrent *active* attack, and finally refer to Lemma 4.9 for the active security of the protocol.

$\mathcal{V}'$ works as follows: when queried by $\mathcal{A}$, it chooses a random challenge a (just like $\mathcal{V}$ does). When $\mathcal{A}$ returns an answer z, j (i.e. the second protocol message), there are two cases to consider:

1. $\mathcal{A}$ previously received answer z', j from $\mathcal{P}$, where $z' = a' \cdot s + e_j$ and a' is $\mathcal{A}$'s query to $\mathcal{P}$. Here we have two possibilities:
   (a) $a = a'$ (which could be the case if $\mathcal{A}$ queried $\mathcal{P}$ during the current protocol execution): in this case, if $z = z'$, $\mathcal{V}'$ accepts; else, it rejects.
   (b) $a \neq a'$: $\mathcal{V}'$ always rejects.
2. $\mathcal{A}$ never previously received an answer of the form z', j from $\mathcal{P}$. In this case $\mathcal{V}'$ always rejects.

Consider the first time $\mathcal{A}$ queries the verifier. The challenge produced by $\mathcal{V}$ has exactly the same distribution as the one $\mathcal{V}'$ outputs. Now, in case 1a, notice that $\mathcal{V}$ will accept if and only if z has the correct value $a \cdot s + e_j$: so $\mathcal{V}'$ always makes the same decision as $\mathcal{V}$. In case 1b, $\mathcal{V}$ rejects except with negligible probability, so $\mathcal{V}'$ is statistically close to the right behavior. This is because accepting would imply that $z = a \cdot s + e_j$, but we also have $z' = a' \cdot s + e_j$ so then $s = (z - z')(a - a')^{-1}$. This happens with negligible probability because s is random and z, z', a, a' are all independent of s. This holds because all of $\mathcal{P}$'s responses (including z') are independent of s. Moreover, since this is the first query, $\mathcal{V}$ has not even looked at s yet, so a, a' and z must be independent of s too. Finally, in case 2, note that no one sees e_j before the adversary produces z, j. If $\mathcal{V}$ accepts, we have $z = a \cdot s + e_j$, so $e_j = z - a \cdot s$, which happens with negligible probability since z, a and s are independent of e_j.

Therefore, we can modify $\mathcal{V}$ so that it behaves like $\mathcal{V}'$ for the first query, and the adversary's advantage changes at most negligibly as a result. Repeating the same argument for all the queries, we reach the game where $\mathcal{V}$ is entirely replaced by $\mathcal{V}'$, and the adversary's advantage is still at most negligibly different from in the original game.

Since $\mathcal{V}'$ does not possess any secret information, an adversary can run $\mathcal{V}'$ "in his head". So for any adversary $\mathcal{A}$ which has non-negligible advantage in a man-in-the-middle attack, we can construct an adversary $\mathcal{A}''$ that emulates both $\mathcal{A}$ and $\mathcal{V}'$ "in his head" and achieves the same advantage, but conducting an *active* attack (since he need not interact with the real verifier $\mathcal{V}$).

Finally, the security of the protocol against concurrent man-in-the-middle attacks follows from the security against concurrent active attacks, which was shown in Lemma 4.9. □

C Proof of PRF security for any polynomial number of queries

In this section we give proofs the security of the function families $\mathcal{F}^{(g,\ell,\mathcal{H})}$ and $\widetilde{\mathcal{F}}^{(g,\ell,\mathcal{H})}$ in general (that is, for any polynomial number of queries).

Theorem C.1. *If g is a PRG with public randomizer that satisfies Property 5.5, then $\mathcal{F}^{(g,\ell,\mathcal{H})}$ is a secure WPRF.*

Proof. As in the proof of Theorem 5.2, since g is a PRG with public randomizer, we can modify the oracle $\mathcal{O}^{F_K}$ (from Definition 2.8) so that the x_s and h_s are uniformly chosen, and the adversary's advantage can increase by at most a negligible amount as a result of this modification. Let $a_1, \ldots, a_k$ be the (uniformly random) inputs to the oracle. Then the distribution of the hashes $h_{s'}(x + a_i)$ in the PRF outputs $g(a_i, h_{s'}(x + a_i))$ as defined in the construction of $\mathcal{F}^{(g,\ell,\mathcal{H})}$ is exactly according to $\mathcal{D}_\ell^n(a_1, \ldots, a_k)$. Then since g satisfies Property 5.5, the outputs of $\mathcal{O}^{F_K}$ must be indistinguishable from the outputs of $\mathcal{O}^{\mathcal{R}}$ where $\mathcal{R}$ is a random oracle. □

Theorem C.2. *Let $\widetilde{\mathcal{F}}^{(g,\ell,\mathcal{H})} = \{\widetilde{F}_s\}$ be the function family obtained by applying the [NR99] technique to the function family $\mathcal{F}^{(g,\ell,\mathcal{H})}$ defined above, for parameters $n, \ell \in \mathbb{Z}$ and $\mathcal{H}$ a family of 2ℓ-wise independent hash functions mapping $(\ell+1)\cdot n$ bits to n bits. Then, if g is a PRG with public randomizer, $\widetilde{\mathcal{F}}^{(g,\ell,\mathcal{H})}$ is a secure PRF family for any adversary making at most ℓ queries. If g furthermore satisfies Property 5.5, then $\widetilde{\mathcal{F}}^{(g,\ell,\mathcal{H})}$ is a secure PRF family (for any* PPT *adversary).*

Proof. Follows directly from Theorems 5.3, 5.2, and 5.6. □

D Efficient Bernoulli sampling, and LPN-based PRG for any τ

The PRG of [ACPS09] relies on the Bernoulli sampling technique of [AIK08], which is defined only for $\tau = 2^{-t}$ for $t \in \mathbb{Z}_+$.

Lemma D.1 ([AIK08])**.** *There exist positive integers $k > 1$ and $c > 2k$ and a sampling algorithm* Samp *that uses $(k + k/c)n$ random bits and outputs a pair (e, v) whose joint distribution is $2^{-\Omega(n)}$-statistically close to $(\mathsf{Ber}^n_{2^{-k}}, \mathcal{U}(\{0,1\}^{kn}))$. Moreover,* Samp *can be implemented in* NC^0.

The sampling method of [AIK08] is as follows: each bit of the Bernoulli output vector e is obtained by taking the product of t uniformly random input bits; and then to reduce entropy waste, they also apply a *randomness extractor* to the input bits to obtain the pseudorandom part of the output, v. The Bernoulli output is just a product of t and thus computable in NC^0, and [AIK08] constructs an extractor that is also in NC^0 so that the whole sampler is in NC^0.

We now generalize their technique to any real $\tau \in (0, \frac{1}{2}]$. The generalized construction is no longer in NC^0, but has logarithmic depth, which is what we require. For security parameter κ, let τ' be the approximation of τ to κ binary places. Note that to a PPT adversary, $\mathsf{Ber}_\tau \overset{c}{\approx} \mathsf{Ber}_{\tau'}$. Now, we can sample from $\mathsf{Ber}_{\tau'}$ by choosing a random value $t \in \{0,1\}^\kappa$ and outputting 1 if $t \le \tau' \cdot 2^\kappa$, and 0 otherwise (for the purposes of the comparison, t should be interpreted as an integer expressed in binary). Sampling n Bernoulli-distributed bits in this way yields an output distribution which is computationally indistinguishable from the desired Ber^n_τ (and this can be done in logarithmic depth). The randomness extractor of [AIK08] is then applied to the input bits just as in their original construction.

For our protocol constructions, it is useful to generate Bernoulli samples separately from the PRG construction, and derived from pseudorandom – rather than truly random – input. To capture this, we define the following.

Definition D.2. *For security parameter κ, let $\mathsf{BerSamp}^G_{n,\tau} : \{0,1\}^n \to \{0,1\}^n$ be the function that: takes as input $x \in \{0,1\}^n$ (which is truly random), applies the pseudorandom generator G to x to obtain a pseudorandom value $x' \in \{0,1\}^{(k+k/c)n}$ where k, c are the constants promised in Lemma D.1 and $\kappa \le k$, then applies the generalized Bernoulli sampling algorithm of to x' to obtain a pair (e, v) whose distribution is computationally indistinguishable from $(\mathsf{Ber}^n_{2^{-k}}, \mathcal{U}(\{0,1\}^{kn}))$, and finally outputs e.*

E PRG with public randomizer from subset sum

The subset sum (SS) problem is the following: given a set of numbers $\{a_1, \ldots, a_d\} \in \mathbb{Z}_M^d$ and a target number $t \in \mathbb{Z}_M$, find a subset $S \subset [d]$ such that $\sum_{i \in S} a_i = t \mod M$. This problem is widely believed to be hard if the a_i are random (subject to certain conditions on the parameters d, M). Define the *subset sum function* $G^{\mathsf{SubsetSum}}_{d,M} : [M]^d \times \{0,1\}^d$ to be the following:

$$G^{\mathsf{SubsetSum}}_{d,M}(a, S) = \sum_{i \in S} a_i \mod M.$$

Impagliazzo and Naor [IN96] showed that if the subset sum function is hard to invert, i.e. one-way, then it is also a pseudorandom generator.

Theorem E.1 ([IN96])**.** *If the subset sum function* $G_{d,M}^{\mathsf{SubsetSum}}(a, S)$ *is one-way, then*

$$\left(a, G_{d,M}^{\mathsf{SubsetSum}}(a, S)\right) \overset{c}{\approx} (a, u)\,,$$

where $a \in \mathbb{Z}_M^d, S \subset \{1, \ldots, d\}$*, and* $u \in \mathbb{Z}_M$ *are chosen uniformly at random.*

This is a PRG with public randomizer, so as in Section 5.3.1, it can be used to *directly* instantiate the function g in the construction of Protocol 4. Also in this case, applying this technique has the advantage that the assumption that g has Property 5.5 corresponds to a weaker assumption (on the hardness of the subset sum problem) than if $G_{d,M}^{\mathsf{SubsetSum}}$ were used as a black-box PRG. Thus we obtain an authentication protocol which is secure against ℓ-concurrent MIM attacks if the subset sum problem is hard, and is furthermore fully secure against concurrent MIM attacks if the subset sum problem is hard even when the input subset S is chosen by a 2ℓ-independent hash function.