Ubiquitous Authentication: Definitions, Attacks, and Constructions

Ivan Damgård[1] and Sunoo Park[2]

[1] Aarhus University
[2] MIT

Abstract. We propose a new approach to the construction of provably secure secret-key authentication protocols for lightweight pervasive devices. We specify a flexible model that captures existing security definitions and can be extended modularly to capture additional protocol features or adversarial capabilities for specific application requirements. Our model allows for *statefulness*, which was not captured by standard definitions in the literature, but is common (and often preferred, due to leakage concerns) in practice. In addition, our model formalizes robustness against *memory erasure attacks*, a type of tampering attack that is not considered by existing theoretical models, yet has been shown to be eminently feasible on many low-cost devices. We show that most existing protocols based on the Learning Parity with Noise (LPN) problem are very much susceptible to memory erasure attacks.
We then propose a two-message protocol in the prover-stateful model which achieves provable *concurrent man-in-the-middle security* and *asymptotically optimal efficiency*. Our protocol can be instantiated based on any secure pseudorandom generator (PRG). We also show a variant construction which is resistant to erasure attacks by using a "reinforced" PRG (that can be built from any PRG), at a cost in key size. Preliminary analysis of our protocol's expected performance when instantiated based on a hardware-oriented stream cipher (Grain) shows that our prover computation time compares favorably against a simple AES-based protocol.

1 Introduction

Secret-key authentication is one of the most basic cryptographic tasks. The setting is that a prover, Peggy, and a verifier, Victor, share a secret key K, and the aim is to design a protocol so that Peggy (and Peggy alone) can convince the verifier that she indeed knows the key K. In this paper, we focus our attention on *one-sided authentication*, as distinct from mutual authentication where *both* parties must be convinced of each other's identity.

Applications demanding efficient one-sided authentication have proliferated in recent years, with the development of technology both affordable and lightweight enough to embed in everyday devices and objects, e.g., in the so-called Internet of Things (IoT). More often than not, such devices started off without sophisticated authentication techniques – or indeed, without any authentication at

all[3] – but attacks quickly emerged against these schemes: ranging from inadequately secured, ubiquitous RFID chips (e.g., building access badges or subway tokens) [5,10,6], to common IoT protocols whose vulnerabilities allow for wireless hacking of "smart home" devices [39], and to remote hijacking of cars by exploiting unsecured communications between car parts [26,31]. These are just a few examples, and the literature contains many more.

Fortunately, this saga has led to a general recognition that having secure authentication even for the smallest system components is important, by both system/policy designers and the public. For example, the car hacking results have led to draft legislation (e.g., in the US Senate [30]) on vehicular cybersecurity standards, as well as dramatic media coverage by a Wired reporter who (consensually) had his car stopped remotely by researchers while he was on the highway [18].

In a certain sense, cryptographers could consider secret-key authentication to be a long solved problem. A simple two-round protocol where the verifier sends a random challenge and the prover replies with a MAC on that challenge turns out to achieve security against concurrent man-in-the-middle attacks, the strongest security definition in the literature. However, authentication on lightweight devices presents interesting challenges *not* addressed by classic solutions like the above. Typically, the prover in these settings is a very resource-limited device such as an RFID chip. Yet efficiency is often of critical importance, whether because busy people expect to speed through the subway turnstile or store checkout without having to wait multiple seconds to authenticate, or because the device in question is part of a pacemaker or automobile where small delays can have devastating consequences.

Accordingly, *lightweight* authentication has lately been a subject of interest. Provably secure lightweight protocols have garnered significant interest, and a number of such protocols have been proposed, in a line of work starting with the HB/HB+ protocols [21,23]. A series of security definitions ranging from *passive* to *concurrent man-in-the-middle* security have become established, and proposed protocols in the theoretical cryptography literature have variously been proven to satisfy one or the other of these security notions. The first formal treatment of secure authentication, due to [2], addressed the setting of *mutual*, rather than *one-sided*, authentication; the security notions currently in the one-sided authentication literature are closely inspired by their seminal definitions.

Now, with the combination of advances both in protocol proposals and the capabilities of low-cost RFID chips, such provably secure protocols are now efficient enough to be within the range of *possible* to be run on existing RFID devices (albeit still not in the lowest cost ranges). So, are we approaching an idealistic union of theory and practice? Not quite. First, the protocols are still

[3] Probably the most pervasive RFID devices are EPC tags, which serve as low-cost electronic barcodes in supply chains and other applications. These often "promiscuously broadcast a static identifier with no explicit authentication procedure" (quoting [23]). The cheapest such tags cost less than $0.01 per unit, and may simply lack adequate resources to do standard cryptographic operations.

far from competitive with deployed solutions in terms of efficiency. This, we may hope to improve over time. Second, and more fundamentally: although the security definitions in the literature protect against strong adversaries (e.g., which can run man-in-the-middle attacks on polynomially many concurrent protocol executions), the existing security notions are based on established adversarial models dating back to the early years of cryptographic theory, which (though they have proven robust and versatile in general) were *not* tailored for the setting of low-cost ubiquitous computing.

As the IoT matures, and RFID and similar technologies permeate more and more aspects of everyday life, a specific new set of requirements for "authentication in ubiquitous computing" is coming into definition. A theoretical model and solutions that address these particular requirements are essential if provable security is ultimately to have relevance to practice. We use the term *ubiquitous authentication* to refer to the problem of one-sided authentication using low-cost pervasive devices.

1.1 Secure authentication: more background and motivation

The very first solution to the authentication problem is due to Goldreich, Goldwasser, and Micali [16], as follows. Suppose we are given a pseudorandom function family (PRF) $\mathcal{F}$ whose functions f_K have a key K and input/output size n bits. The crucial property of a PRF is that even an adversary who chooses the input (but does not know the key) cannot distinguish the output of f_K from random. Given a PRF, we can simply let the verifier send a random input x, and have the prover respond with $f_K(x)$. This simple two-round protocol already achieves concurrent MIM security[4].

Asymptotic efficiency. From the perspective of efficiency, this simple solution is problematic in general. The standard construction of a PRF (from [16]) is based on a pseudorandom generator (PRG), which is a much simpler primitive that expands a short key into a random-looking longer output. However, the GGM construction does not yield a particularly efficient PRF: even assuming a very efficient PRG that only requires a constant number of operations per output bit, the PRF will have complexity $\Omega(n^2)$.

There are in fact several very efficient and low-depth constructions of PRGs from specific problems, such as Learning Parity with Noise (LPN); and furthermore, even linear-time computable PRGs with linear stretch are known to follow from general and plausible assumptions[5]. Hence, it is compelling to try to de-

[4] Note that for authentication, it is actually sufficient to have an unpredictable function, a computationally secure "MAC", rather than a PRF. However, we do not know of more efficient constructions of MACs either, so this does not help in terms of efficiency. In [22], Ishai et al. construct constant-overhead PRFs and MACs, but in their work the output size is much smaller than the input: in our context, this would make the prover's answer much easier to guess than the verifier's challenge, making the protocol insecure.

[5] A number of different sufficient conditions for such PRGs are known. In [22] it is observed that such PRGs follow from Alekhnovich's variant of the Learning Parity

sign authentication directly based on weaker pseudorandom primitives such as PRGs, in such a way that the protocol inherits the efficiency. More specifically, we direct our attention to the open question of designing a two-round protocol to achieve (optimal) linear computational complexity.

Practical efficiency. In practice, a natural way to implement the simple PRF-based authentication protocol described above would be to use a block cipher such as AES in place of the PRF. While widely-used block ciphers such as AES are highly streamlined, we observe that (synchronous) *stream ciphers* can be used as PRGs and are often much faster than block ciphers. Thus, a natural question arises: can we design authentication directly based on stream ciphers, in such a way that the protocol inherits the efficiency? Efficiency gains at this scale could be particularly significant in settings where the prover is a lightweight device. Indeed, building authentication based on stream ciphers has been recognised as a promising topic in the RFID authentication literature, for exactly this reason [14]; however, to the best of our knowledge, concrete proposals so far have required three or more rounds and have security which is (if proved at all) based upon non-standard assumptions that are stronger than assuming the stream cipher is a PRG (e.g. [27,4]).

The problem of optimally efficient two-round authentication has been long open in the (standard) stateless model. This paper can be seen to ask, and affirmatively answer, the following question: *can we leverage a small amount of state on the prover side to build far more efficient protocols?*

1.2 Overview of contributions

In this paper, we propose a new, flexible theoretical model for ubiquitous authentication. Our framework captures the full range of above-mentioned security definitions that are already considered in the literature, and also incorporates additional parameters that better capture realistic considerations when dealing with low-cost devices, both in terms of the capabilities of the adversary and of the lightweight prover device. Notably, our model allows for statefulness of the prover and/or verifier, which was not captured by standard definitions in the literature, but is common (and often preferred, due to leakage concerns) in practice. Our model incorporates the accompanying issues of synchronisation and resetting which are not addressed by the existing stateless models.

Furthermore, we consider robustness against *memory erasure attacks*, a type of realistic tampering attack that is not covered by the standard theoretical models.[6] Our definition formalizes the natural requirement that though an ad-

with Noise assumption. Applebaum [1] shows that such PRGs can be obtained from the assumption that a natural variant of Goldreich's candidate for a one-way function in NC0 is indeed one-way. The improved HILL-style result of Vadhan and Zheng [38] implies that such PRGs can be obtained from any exponentially strong one-way function that can be computed by a linear-size circuit.

[6] The memory on low-cost RFID tags is not well-protected, and the ubiquity of their usage means that it is often relatively easy for an attacker to obtain (temporary)

versary may disable a prover device by erasing its memory, the adversary must not thereby learn secret information.

We then show that a number of existing protocols based on the Learning Parity with Noise (LPN) problem – namely, [21,23,25] – which have been proven secure under the traditional security definitions, are very much susceptible to erasure attacks (we give attacks that allow complete recovery of the secret key).

Next, we construct a two-message protocol in the prover-stateful model, called UbiAuth, which can be instantiated with any secure pseudorandom generator (PRG). We emphasize that in the prover-stateful model we do *not* have to deal with synchrony issues between the prover and verifier as only the prover needs to keep state. We prove that UbiAuth achieves concurrent man-in-the-middle security. Moreover, a slight variant protocol instantiated with a "reinforced" PRG is robust against erasure attacks, at the cost of an $n^{1+\varepsilon}$ factor increase in key size. We give a construction of such reinforced PRGs based on any secure PRG. Additionally, we prove that a further variant protocol, UbiAuth+, provides some (*a priori* bounded) resilience against resetting attacks. The two variants can be combined to achieve security against both erasure attacks and bounded resetting attacks.

A feature of interest from a theoretical standpoint is that our protocols are the first concurrent man-in-the-middle secure protocols to achieve *amortised linear time complexity* (concretely, less than 1.5 linear-time PRG calls per protocol execution). A more detailed discussion of how our protocols relate to existing protocols in the literature is given in Section 2.

Finally, returning to the motivation of theory in the context of practice, we consider how UbiAuth would perform if instantiated with a stream cipher acting as the PRG. Specifically, a preliminary analysis of expected performance of our protocol instantiated with Grain-128 yields a favorable comparison with a simple AES-based protocol[7] in terms of prover computation time.

In summary, the rest of the paper is structured as follows.

- Flexible new model designed for ubiquitous authentication (subsuming prior definitions), incorporating statefulness and erasure attacks, and allowing for modular extension to model specific adversarial capabilities. (Section 3.)
- Erasure attacks against existing LPN-based lightweight authentication protocols. (Section 4.)
- Two-message protocols achieving linear time complexity and provable security based on any PRG, in the prover-stateful model. (Section 5.)
- Preliminary performance analysis yields favorable comparison with simple AES-based protocol. (Section 6.)

---

physical access to the device. A particular type of attack which has been shown to be feasible at relatively low cost on the EPROM memory used in RFID tags is erasing at specific locations using UV light: [37] describes a method using everyday supplies such as an $8 laser pointer.

[7] AES itself is too heavyweight for the very lightweight applications we are ultimately targeting, so this is not intended to indicate that our protocol as is might be suitable for low-cost EPC tags, but rather to indicate the reasonableness of our protocol as a prototype to be improved upon.

2 Related work

LPN-based protocols. The first and simplest authentication scheme based on LPN was the HB scheme [21], which is secure against passive attacks (but easily breakable by active attacks). Subsequently, [23] gave an actively secure[8] variant protocol called HB+ with an additional round (which requires the prover to keep state between rounds). Then, [25] proposed the first two-round actively secure protocol, whose security is based on the Subspace LPN problem;[9] and shortly thereafter, [20] proposed the "Lapin" protocol, a more efficient two-round protocol whose security is based on another LPN variant called Ring LPN.

The preceding protocols are all vulnerable to man-in-the-middle attacks. [25] 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, [13] gave a more efficient MIM secure (three-round) variant protocol using more efficient MACs that make use of pairwise independent hashing. We note that both of these constructions suffer from large (quadratic) MAC key-sizes.

Protocols from abstract primitives. Dodis et al. [13] construct a three-round protocol based on any weak PRF. A weak PRF is a relaxed notion of PRF in which outputs are only required to look random for random (rather than adversarial) inputs. [13] achieves active security (a weaker notion than MIM security). Subsequently, Lyubashevsky and Masny [28] proposed a three-message protocol based on any weak PRF, which is secure against sequential MIM attacks[10]. In addition, they give a variant three-message protocol that can be built from any *randomised* weak PRF, a yet slightly weaker primitive. However, these protocols are not concurrent MIM secure (in fact, [28] outlines an attack), and all require three messages.

Recent work by Cash, Kiltz, and Tessaro [8] (which is subsequent to but independent of our work) proposes sequential MIM secure two-round authentication by a generic transformation from actively-secure protocols which satisfy certain requirements. Their construction can be instantiated from a number of existing protocols, including ones based on concrete assumptions such as LPN or the decision Diffie-Hellman assumption, and also based on weak PRFs. However, [8] do not show concurrent MIM security.

[8] In fact, it was subsequently shown that HB+ is secure even against *concurrent active* attacks where the adversary may interact with multiple provers concurrently [24].

[9] Subspace LPN is a variant of the LPN problem that has been shown to be almost as secure as standard LPN, under certain conditions [33].

[10] Sequential MIM security is stronger than active security, but weaker than concurrent MIM security. In a sequential MIM attack, the adversary can interact polynomially many times with an honest prover and verifier, but the interactions cannot involve multiple concurrent sessions with the prover (or the verifier). In contrast, an active adversary can interact only with the prover.

3 Defining security

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 ACC or REJ – this is also called the *output* of the protocol execution.

The usual definition of a secret-key authentication, where the prover and verifier are *stateless* (i.e., do not maintain state between protocol executions), is given formally in Definition 1 below.

Notation. For a finite set B, we write $b \leftarrow B$ to denote that $b \in B$ is sampled uniformly at random from B. For $n \in \mathbb{N}$, $[n]$ denotes the set $\{1, 2, \ldots, n\}$. An *efficient* algorithm is one which runs in probabilistic polynomial time (PPT).

Definition 1. *A **stateless secret-key authentication protocol** is a tuple $(\mathsf{Gen}, \mathcal{P}, \mathcal{V})$ where:*

- Gen *is a* PPT *key-generation algorithm, which takes as input the security parameter 1^κ (in unary) and outputs a secret key s.*
- $\mathcal{P}$ *is an interactive Turing machine representing the prover, which takes as input 1^κ, a secret key s and outputs (interactively) the messages sent by the prover.*
- $\mathcal{V}$ *is an interactive Turing machine representing the verifier, which takes as input 1^κ, a secret key s and outputs (interactively) the messages sent by the verifier. On completion of a protocol execution, $\mathcal{V}$ outputs* ACC *or* REJ*.*

Note that for brevity, we sometimes omit writing the security parameter 1^κ as input. For the standard security definitions, we refer to Appendix A; due to space constraints, we give here detailed security definitions only for the new, stateful case (below).

3.1 Fully stateful protocols

In Definition 2, we extend Definition 1 to allow for the prover and verifier keeping persistent state between protocol executions.

Definition 2. *A **fully stateful secret-key authentication protocol** is a tuple $(\mathsf{Gen}, \mathcal{P}, \mathcal{V})$ where:*

- Gen *is a* PPT *key-generation algorithm, which takes as input the security parameter 1^κ (in unary) and outputs a secret key s and an initial pair of states (σ_0, τ_0).*
- $\mathcal{P}$ *is an interactive Turing machine representing the prover, which takes as input 1^κ, a secret key s and a state σ, and outputs (interactively) the messages sent by the prover. Upon conclusion of a protocol execution, $\mathcal{P}$ outputs an updated state σ', denoted $\sigma' \leftarrow \mathcal{P}(s, \sigma)$.*

– $\mathcal{V}$ *is an interactive Turing machine representing the verifier, which takes as input* 1^κ*, a secret key* s *and a state* τ*, and outputs (interactively) the messages sent by the verifier. Upon conclusion of a protocol execution,* $\mathcal{V}$ *outputs* (τ', b) *where* τ' *is an updated state and* $b \in \{\text{ACC}, \text{REJ}\}$.

For any $i \in \mathbb{N}$, we define $\mathcal{P}^i(s, \cdot)$ to be the i-fold composition of $\mathcal{P}(s, \cdot)$ with itself. We define $\mathcal{V}^i(s, \cdot)$ analogously.

Definition 3. *A fully stateful authentication protocol is* ***complete*** *if there is a negligible function* ε *such that for any polynomial* p*,*

$$\Pr_{(s,\sigma_0,\tau_0)\leftarrow\mathsf{Gen}(1^\kappa)}\left[(\mathcal{P}^{p(\kappa)}(s,\sigma_0),\mathcal{V}^{p(\kappa)}(s,\tau_0)) = \text{REJ}\right] \leq \varepsilon(\kappa) \ .$$

Active security. An authentication protocol is secure against *active attacks* if for any adversary 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 $\mathcal{V}$, the probability that $\mathcal{V}$ accepts is negligible. This is formalized in Definition 4.

Definition 4. *A fully stateful authentication protocol* $(\mathsf{Gen}, \mathcal{P}, \mathcal{V})$ *is* ***secure against active attacks*** *if there is a negligible function* ε *such that for any (two-part)* PPT *adversary* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$*,*

$$\Pr\left[\begin{array}{l}(s,\sigma_0,\tau_0) \leftarrow \mathsf{Gen}(1^\kappa) \\ \omega \leftarrow \mathcal{A}_1^{\widehat{\mathcal{P}}(s,\sigma_0,\cdot)}(1^\kappa) \\ B \leftarrow (\mathcal{A}_2(\omega), \mathcal{V}(s,\tau_0))\end{array} : B = \text{ACC}\right] \leq \varepsilon(\kappa) \ ,$$

where the stateful oracle $\widehat{\mathcal{P}}(s, \sigma_0, \cdot)$ *is an interactive Turing machine that runs* $\mathcal{P}$ *on the state outputted by the previous invocation of* $\mathcal{P}$ *(or on* σ_0 *for the first invocation), and outputs the messages outputted by* $\mathcal{P}$*.*

Concurrent MIM security. An authentication protocol is secure against *concurrent man-in-the-middle (MIM) attacks* if for any adversary which first can interact arbitrarily polynomially many times with an honest prover $\mathcal{P}$ and/or an honest verifier $\mathcal{V}$ (the interactions may be concurrent, but the adversary cannot reset the prover's state), and then afterward (now, without access to $\mathcal{P}, \mathcal{V}$) interacts once with an honest verifier $\mathcal{V}'$, the probability that $\mathcal{V}'$ accepts is negligible. This is formalized in Definition 5.

Definition 5. *A fully stateful authentication protocol* $(\mathsf{Gen}, \mathcal{P}, \mathcal{V})$ *is* ***secure against concurrent MIM attacks*** *if there is a negligible function* ε *such that for any* PPT *adversary* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$*,*

$$\Pr\left[\begin{array}{l}(s,\sigma_0,\tau_0) \leftarrow \mathsf{Gen}(1^\kappa) \\ \omega \leftarrow \mathcal{A}_1^{\widehat{\mathcal{P}}(s,\sigma_0,\cdot),\widehat{\mathcal{V}}(s,\tau_0,\cdot)}(1^\kappa) \\ B \leftarrow (\mathcal{A}_2(\omega), \mathcal{V}(s,\tau_0))\end{array} : B = \text{ACC}\right] \leq \varepsilon(\kappa) \ ,$$

where the stateful oracles $\widehat{\mathcal{P}}(s, \sigma_0, \cdot)$ and $\widehat{\mathcal{V}}(s, \tau_0, \cdot)$ are interactive Turing machines that run $\mathcal{P}$ and $\mathcal{V}$ on the state outputted by the previous invocation, respectively (or on σ_0, τ_0 for the first invocations), and outputs the messages outputted by $\mathcal{P}$ and $\mathcal{V}$ (including $\mathcal{V}$'s ACC/REJ *decision).*

ℓ-instance concurrent MIM security. Concurrent MIM security is the strongest security definition in the literature for stateless protocols. We now define an even stronger notion relevant to the stateful setting, where the adversary has access to many copies or clones of the honest prover/verifier. Note that this definition covers the case of an adversary who can reset the parties' states: clearly, an adversary who resets ℓ times can be emulated by an adversary having access to ℓ copies of the prover/verifier.

An authentication protocol is secure against *unbounded-instance concurrent man-in-the-middle attacks* if for any adversary which first can interact arbitrarily polynomially many times with polynomially many honest provers $\mathcal{P}_1, \ldots, \mathcal{P}_k$ and/or honest verifiers $\mathcal{V}_1, \ldots, \mathcal{V}_k$, and then afterward (now, without access to the $\mathcal{P}_i, \mathcal{V}_i$) interacts once with an honest verifier $\mathcal{V}'$, the probability that $\mathcal{V}'$ accepts is negligible. This is formalized in Definition 6.

Definition 6. *A fully stateful authentication protocol $(\mathsf{Gen}, \mathcal{P}, \mathcal{V})$ is **secure against unbounded-instance concurrent MIM attacks** if there is a negligible function ε such that for any* PPT *adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$, and any polynomial $p = p(\kappa)$,*

$$\Pr\left[\begin{array}{l}(s, \sigma_0, \tau_0) \leftarrow \mathsf{Gen}(1^\kappa) \\ \omega \leftarrow \mathcal{A}_1^{\{\widehat{\mathcal{P}}(s,\sigma_0,\cdot)\}^p, \{\widehat{\mathcal{V}}(s,\tau_0,\cdot)\}^p}(1^\kappa) \\ B \leftarrow (\mathcal{A}_2(\omega), \mathcal{V}(s, \tau_0))\end{array} \;:\; B = \text{ACC}\right] \leq \varepsilon(\kappa) \ ,$$

where the stateful oracles $\widehat{\mathcal{P}}(s, \sigma_0, \cdot)$ and $\widehat{\mathcal{V}}(s, \tau_0, \cdot)$ are as in Definition 5, and the notation $\mathcal{A}_1^{\{\mathcal{O}\}^p}$ means that $\mathcal{A}_1$ has oracle access to p independent copies of oracle $\mathcal{O}$.

A natural relaxation of unbounded-instance concurrent MIM security addresses the case when the adversary has access to only ℓ provers and verifiers, where $\ell = \mathsf{poly}(n)$ is bounded in advance. We call this security notion ***ℓ-instance concurrent MIM security***. This definition allows for constructions to take advantage of the fact that ℓ is chosen and fixed initially. We believe this would be a more realistic attack model in many scenarios, since it may be practically infeasible for an adversary to obtain arbitrarily many clones, or to interact with arbitrarily many provers concurrently: for example, with passive RFID tags, the physical range of communication is very short, and limits the number of simultaneous interactions to a small radius per adversarial device; and the number of possible resets to an RFID tag's state is realistically very much bounded by the number of writes that its memory can support.

Furthermore, we can define ***unbounded-instance active security*** and ***ℓ-instance active security*** to be the corresponding notions for active security:

that is, where the adversary has access to polynomially many (or up to ℓ, respectively) honest provers.

Synchrony considerations. A potential drawback of stateful protocols is maintaining synchronisation of the the prover's and verifier's states. First, we highlight that *this is not always necessary*: for example, a protocol might work as long as the prover and verifier each update their own state at every execution (e.g., in order not reuse randomness), but *not* require any synchronisation between the parties.

If there *is* a notion of synchrony between the two parties' states, it is important to note that their states may become out of sync very easily: whether by accident, or adversarially engineered circumstances, or even in normal operation if there are multiple provers/verifiers rather than just one pair. It is therefore unacceptable if lack of synchrony causes

1. security vulnerabilities, or
2. substantial disruption to operation.

Bullet (1) is already addressed by the security definitions above, which guarantee that the adversary cannot falsely authenticate even when given access to prover(s) and verifier(s) which are out of sync.[11]

Bullet (2) can be addressed by building into the protocol a resynchronisation procedure to be performed if synchrony is lost. If the initial states σ_0, τ_0 are stored by the prover and verifier (respectively), then "recovery" is always possible by resetting the states to σ_0, τ_0. If the protocol is secure against unbounded-instance concurrent MIM attacks, this could be an acceptable resynchronisation method. In general, a fully stateful protocol where synchrony matters should be equipped with an interactive resynchronisation protocol $(\mathsf{Resync}_{\mathcal{P}}, \mathsf{Resync}_{\mathcal{V}})$ using which the parties can regain synchrony if out of sync.

Definition 7. *Let $(\mathsf{Gen}, \mathcal{P}, \mathcal{V})$ be a fully stateful authentication protocol. A **resynchronisation protocol** $(\mathsf{Resync}_{\mathcal{P}}, \mathsf{Resync}_{\mathcal{V}})$ is a tuple of interactive Turing machines as follows.*

- $\mathsf{Resync}_{\mathcal{P}}$ *takes as input 1^κ, a secret key s, and a state σ, and outputs a new state σ'.*
- $\mathsf{Resync}_{\mathcal{V}}$ *takes as input 1^κ, a secret key s, and a state τ, and outputs a new state τ'.*

For a bit-string $\mathbf{b} = (b_1, \ldots, b_p) \in \{0,1\}^p$, define $\mathcal{P}^{\mathbf{b}}(s, \cdot)$ as

$$\mathcal{P}'_{b_p}(s, \mathcal{P}'_{b_{p-1}}(s, \ldots \mathcal{P}'_{b_1}(s, \cdot) \ldots)) \text{ , where}$$
$$\mathcal{P}'_b(s, \cdot) = \begin{cases} \mathcal{P}(s, \cdot) & \text{if } b = 0 \\ \mathsf{Resync}_{\mathcal{P}}(s, \cdot) & \text{if } b = 1 \end{cases} .$$

[11] The oracles $\widehat{\mathcal{P}}$ and $\widehat{\mathcal{V}}$ are designed to make sure that the adversary only gets access to $\mathcal{P}$ and $\mathcal{V}$ with properly updated state. However, the prover and verifier oracles are *not* synchronised with each other, so the security definitions directly give the desired guarantee that the adversary's advantage is negligible even when given access to unsynchronised prover(s) and verifier(s).

Let $\mathcal{V}^{\mathbf{b}}(s,\cdot)$ *be defined analogously.*

We say that $(\mathsf{Resync}_{\mathcal{P}}, \mathsf{Resync}_{\mathcal{V}})$ *is a* ***complete resynchronisation protocol for*** $(\mathsf{Gen}, \mathcal{P}, \mathcal{V})$ *if there is a negligible function* ε *such that for any polynomial* $p = p(\kappa)$ *and any* $\mathbf{b} \in \{0,1\}^p$*,*

$$\Pr_{(s,\sigma_0,\tau_0)\leftarrow\mathsf{Gen}(1^\kappa)}\left[(\mathcal{P}^{\mathbf{b}}(s,\sigma_0),\mathcal{V}^{\mathbf{b}}(s,\tau_0)) = \text{REJ}\right] \leq \varepsilon(\kappa) \ .$$

Of course, we need to impose some security requirements on the resynchronisation protocol: resynchronising should not reveal any secret information to the adversary. The security requirements are detailed in Section 3.2.

Prover-stateful protocols. We emphasise that for the constructions in this paper, synchrony is not a problem, because our protocols will be in the *prover-stateful model* where only the prover needs to maintain persistent state. In our constructions, the verifier can *optionally* maintain state to improve its efficiency.

In general, we would advocate the prover-stateful model over the fully stateful model, as it allows to take advantage of statefulness for protocol design, while straightforwardly avoiding synchrony problems. It is nonetheless valuable to have definitions covering relevant synchrony considerations for the fully stateful model, both for completeness and because fully stateful variants of prover-stateful protocols – like ours – may have efficiency advantages (while still straightforwardly avoiding synchrony problems).

3.2 Extending the attack model

We have designed the security definitions to allow for easy extensions, in cases where we want to model additional protocol features or adversarial capabilities. We believe that this kind of extensible definitional design is particularly important for "forward-compatibility" in settings where technology is fast-changing and/or it may be desirable to optimise protocol designs for application-specific security requirements. By using a unified language to express security requirements across different cases, comparisons between security notions can be more easily and fairly done, and security proofs in one setting can be more readily transferred to other settings (e.g., even if the security requirements in one setting are more demanding than in another, a weaker security proof might be used as a building block in a stronger one, if there is sufficient common ground between definitions). Moreover, features of definitions can be combined a modular way, allowing mixing and matching of security requirements to target particular applications.

We now give two examples of useful extensions to the definitions given in Section 3.1. Both are implemented by changing the behaviour of the oracles $\widehat{\mathcal{P}}$ and $\widehat{\mathcal{V}}$ used in the security definitions. The first is formalizing security for an *additional protocol feature*, namely, the resynchronisation protocols discussed above. The second is modeling an *additional adversarial capability*, namely, the potential for "erasure attacks", which will be described in detail in Section 3.2.

Resynchronisation protocols

Resynchronisation protocols have already been defined in Section 3.1. We now give a security definition which captures the requirement that an adversary should not learn any secret information from the resynchronisation protocol.

Definition 8. *Let* $(\mathsf{Gen}, \mathcal{P}, \mathcal{V})$ *be an authentication protocol, and let* $(\mathsf{Resync}_{\mathcal{P}}, \mathsf{Resync}_{\mathcal{V}})$ *be a complete resynchronisation protocol for* $(\mathsf{Gen}, \mathcal{P}, \mathcal{V})$*. Let* $\widehat{\mathcal{P}}_{\mathsf{resync}}(s, \cdot)$ *and* $\widehat{\mathcal{V}}_{\mathsf{resync}}(s, \cdot)$ *be defined exactly like* $\widehat{\mathcal{P}}(s, \cdot)$ *and* $\widehat{\mathcal{V}}(s, \cdot)$ *from Definition 5, but with the additional feature that on a special input* resynchronise*, they update the state of the prover (or verifier) by running* $\mathsf{Resync}_{\mathcal{P}}(s, \cdot)$ *(or* $\mathsf{Resync}_{\mathcal{V}}(s, \cdot)$*, respectively).*

We say that $(\mathsf{Gen}, \mathcal{P}, \mathcal{V}, \mathsf{Resync}_{\mathcal{P}}, \mathsf{Resync}_{\mathcal{V}})$ *is* ***resynchronisable and secure against concurrent MIM attacks*** *if it satisfies* concurrent MIM security *(Definition 5) with respect to newly defined oracles* $\widehat{\mathcal{P}}_{\mathsf{resync}}$ *and* $\widehat{\mathcal{V}}_{\mathsf{resync}}$*.*

Exactly analogous definitions can be made for *active*, *ℓ-instance*, and *unbounded-instance* variants of the security definition, by substituting the corresponding definition into the statement of Definition 8.

Erasure attacks

The memory on low-cost RFID tags is not well-protected, and the ubiquity of their usage means that it is often relatively easy for an attacker to obtain (temporary) physical access to the device. A particular type of attack which has been shown to be feasible at relatively low cost on the EPROM memory used in RFID tags is erasing at specific locations using UV light: [37] describes a method using everyday supplies such as an $8 laser pointer.

It may be inevitable that such attacks can disable a prover device, but a desirable protocol feature would be that an adversary running erasure attacks can *only* disable the prover, and cannot gain the ability to impersonate the prover. This is captured in Definition 9.

Definition 9. *Let* $(\mathsf{Gen}, \mathcal{P}, \mathcal{V})$ *be an authentication protocol. Let* $\widehat{\mathcal{P}}_{\mathsf{erase}}(s, \cdot)$ *be defined exactly like* $\widehat{\mathcal{P}}(s, \cdot)$ *in Definition 5, but with the additional feature that on receiving a special type of input* (erase, i) *for* $i \in \{1, \ldots, |s|\}$*, all future operations are performed using a modified secret key š which is equal to* s *but with the* i*th bit set to* 0*. (Note that the "erase" operation can be performed multiple times to set multiple locations to* 0*.)*

We say that $(\mathsf{Gen}, \mathcal{P}, \mathcal{V})$ *is* secure against concurrent MIM attacks with erasures *if it satisfies* concurrent MIM security *(Definition 5) with respect to verifier oracle* $\widehat{\mathcal{V}}$ *and newly defined prover oracle* $\widehat{\mathcal{P}}_{\mathsf{erase}}$*.*

Again, exactly analogous definitions can be made for *active*, *ℓ-instance*, and *unbounded-instance* variants of the security definition, by substituting the corresponding definition into the statement of Definition 9. Moreover, we remark that Definitions 8 and 9 can be straightforwardly combined to yield a natural definition of *resynchronisable and secure against concurrent MIM attacks with erasures.*

4 Attacks on LPN-based protocols

In this section, we show that a number of proposed protocols in the lightweight authentication literature are vulnerable to erasure attacks: namely, the HB and HB+ protocols [21,23] and the two-round protocol of [25] (henceforth, "the KPCJV protocol"). The security of these protocols (passive, active, and active, respectively) is proven in [21,23,24,25] by reduction to the hardness of the Learning Parity with Noise (LPN) problem (or Subspace LPN, in the case of [25]). However, all of these proofs are with respect to the traditional (stateless) security definitions, which do not consider the possibility of erasure attacks.

4.1 Overview of HB and HB+

Notation. We denote vectors by bold lower-case letters, and matrices by bold upper-case letters. For a vector $\mathbf{v}$, we write v_i to denote its ith coordinate. Arithmetic operations on n-bit vectors are taken over the field of 2^n elements. 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.

Protocol 1. The HB protocol [21]

Public parameters. Security parameter $\kappa \in \mathbb{Z}$, $n, m \in \mathbb{Z}$ polynomial in κ, noise rate $\tau \in (0, \frac{1}{2})$, threshold $\tau' \in (\tau, \frac{1}{2})$.
Key generation. $\mathsf{HB.Gen}(1^\kappa)$ samples secret key $\mathbf{s} \leftarrow \mathbb{Z}_2^n$.

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

We omit the details of the HB+ and KPCJV protocol due to space constraints. That protocol follows a substantially similar structure to the HB and HB+ protocols; in particular, the final message is a collection of LPN samples, and the verifier's decision is based on whether the last message is an LPN sample with below a certain threshold of noise. Due to this shared structure, the KPCJV protocol yields quite straightforwardly to essentially the same attacks we describe for HB and HB+ in the next subsection.

4.2 The erasure attack

We present the attack in the context of HB, since it is the simplest of the protocols.

High-level attack structure. Suppose for a moment that we have a procedure RecoverBit that, for any $i \in [n]$, erases the ith bit of $\mathbf{s}$ (on the prover), performs some computations with oracle access to the (erased) prover and verifier, and outputs the correct value of s_i with overwhelming probability. If we have a "multi-instance" adversary which has access to $\ell \geq n$ copies of the prover, then we are already done, since the adversary can run RecoverBit independently on different copies of the prover to recover all the bits of $\mathbf{s}$ with overwhelming probability.

However, if the adversary has erasure access only to a single copy of the prover, it is slightly more tricky to recover all the bits of $\mathbf{s}$: once we have already performed erasures on the prover, we cannot guarantee that RecoverBit will continue to work for successive erasures. If the bit s_i was originally equal to 0, then there is no problem because the erasure did not change anything. To address the case where s_i was originally equal to 1, we observe that it is possible to *simulate* the original prover after performing RecoverBit, as follows.

Simulating the original prover. For $j \in [m]$, let $\mathbf{a}_j \in \mathbb{Z}_2^n$ be the jth column of $\mathbf{A} \in \mathbb{Z}_2^{n \times m}$, and let $a_{j,i} \in \{0,1\}$ be the ith bit of $\mathbf{a}_j$. Observe that in the HB protocol, the jth bit z_j of the second message $\mathbf{z}$ is equal to $\langle \mathbf{s}, \mathbf{a}_j \rangle + e_j$. The verifier accepts if and only if the number of bits e_j which are nonzero is less than $\tau' \cdot n$. The crucial property of the prover's message $\mathbf{z}$ is that for each $j \in [n]$, $\Pr[z_j = \langle \mathbf{s}, \mathbf{a}_j \rangle] = \tau$.

Consider what happens after we erase the ith bit of $\mathbf{s}$ on the prover, changing s_i from 1 to 0 (recall that we assumed $s_i = 1$ originally). Let $\mathbf{s}$ denote the original secret key, and $\check{\mathbf{s}}$ denote the new secret key with the ith bit changed to 0. For each $j \in [m]$, we have that $\langle \mathbf{s}, \mathbf{a}_j \rangle \neq \langle \check{\mathbf{s}}, \mathbf{a}_j \rangle$ if and only if $a_{j,i} = 0$. It follows that

$$\Pr_{\check{\mathbf{z}} \leftarrow \mathcal{P}(\check{\mathbf{s}}, \mathbf{A})} [\check{z}_j = \langle \mathbf{s}, \mathbf{a}_j \rangle] = \begin{cases} \tau & \text{if } a_{j,i} = 0 \\ 1 - \tau & \text{if } a_{j,i} = 1 \end{cases}.$$

Finally, note that the adversary knows the values $a_{j,i}$ (since $\mathbf{A}$ is sent publicly in the protocol), so can perfectly simulate the distribution of the original prover's message $\mathbf{z}$ (in response to any first message $\mathbf{A}$) by:

1. running the erased prover on $\mathbf{A}$ to obtain $\check{\mathbf{z}}$
2. for each $j \in [m]$, flipping the bit $\check{z}_j$ if $a_{j,i} = 1$.

Moreover, it is clear that this simulation technique can be applied independently for multiple erased bit positions $i \in [n]$. We conclude that the procedure RecoverBit can be used to recover every bit of $\mathbf{s}$ with overwhelming probability, even by an adversary who has access to only one prover, since this simulation technique allows the adversary to simulate a fresh prover even after RecoverBit has been performed.

It now remains to describe the procedure RecoverBit.

Recovering the ith bit of the secret. Our basic strategy will be to erase the ith bit of $\mathbf{s}$ on the prover, and see if the prover/verifier's behaviour changes as a result. If so, then we conclude that s_i was originally 1 and the erasure changed

it to 0. If not, then we conclude that s_i was originally 0, so the erasure did not change anything.

Case 1. Let us first consider the behaviour of the prover before any erasure is performed. By completeness, the probability that the honest verifier accepts the honest prover's response $\mathbf{z}$ is overwhelming.

Case 2. Now let us consider the behaviour of the prover after an erasure which changes s_i from 1 to 0. Let $\mathbf{s}$ denote the original secret key, and $\check{\mathbf{s}}$ denote the new secret key with the ith bit changed to 0. As observed earlier, for each $j \in [m]$, $\langle \mathbf{s}, \mathbf{a}_j \rangle \neq \langle \check{\mathbf{s}}, \mathbf{a}_j \rangle$ iff $a_{j,i} = 0$. Since the honest verifier samples $\mathbf{A}$ randomly, this means the distribution of $\check{\mathbf{z}}$ is

$$\mathbf{sA} + \mathsf{Ber}_{\tau}^{m} + \mathsf{Ber}_{1/2}^{m} = \mathbf{sA} + \mathsf{Ber}_{1/2}^{m} \ .$$

Then the probability that the honest verifier accepts $\check{\mathbf{z}}$ as a response to a uniformly sampled $\mathbf{A}$ is

$$\begin{aligned} \Pr_{\substack{\mathbf{A} \leftarrow \mathbb{Z}_2^{n \times m} \\ \check{\mathbf{z}} \leftarrow \mathcal{P}(\check{\mathbf{s}}, \mathbf{A})}} [||\check{\mathbf{z}} + \mathbf{sA}||_1 < \tau' \cdot m] &= \Pr_{\check{\mathbf{e}} \leftarrow \mathsf{Ber}_{1/2}^{m}} [||\check{\mathbf{e}}||_1 < \tau' \cdot m] \\ &\leq O(\exp(-m)) \ , \end{aligned}$$

where the final inequality follows from a Chernoff bound.

Algorithm 1. RecoverBit

Input. $i \in [n]$, the bit position to recover.
Oracles. $\widehat{\mathcal{P}}_{\mathsf{erase}}(\mathbf{s}, \cdot)$ and $\widehat{\mathcal{V}}(\mathbf{s}, \cdot)$.
1. Send (erase, i) to $\widehat{\mathcal{P}}_{\mathsf{erase}}$.
2. Run a protocol execution between the honest verifier and modified prover.
3. If the verifier accepts, output 0. Else, output 1.

Our analyses of Case 1 and Case 2 imply that the algorithm RecoverBit, specified below, outputs the correct value of s_i with overwhelming probability. This is stated more formally in Theorem 1.

Theorem 1. *There is a negligible function ε such that for all $\kappa \in \mathbb{N}$, for all $i \in [n]$ where $n = n(\kappa)$ is defined as in the HB protocol,*

$$\Pr_{\mathbf{s} \leftarrow \mathsf{HB.Gen}(1^{\kappa})} [\mathsf{RecoverBit}(i) = s_i] \geq 1 - \varepsilon(\kappa) \ .$$

In summary: we have described and proven the efficacy of an adversary against the HB protocol which, given erasure access to a single copy of the honest prover and oracle access to the honest verifier, fully recovers the secret key with overwhelming probability, by using RecoverBit on each bit position $i \in [n]$ and using the simulation procedure described earlier in this section to emulate the original prover's behaviour before erasure.

Theorem 2. *There is a* PPT *adversary against the HB protocol which, given erasure access to a single copy of the honest prover and oracle access to the honest verifier, outputs the secret key with overwhelming probability.*

Discussion. The described attack can be straightforwardly extended for the HB+ and KPCJV protocols. We emphasise that this attack lies outside the model in which the security of these protocols was originally proven by their authors.

Interestingly, of the LPN-based lightweight authentication protocols in the literature, the Lapin protocol [20] is the notable one which does not obviously succumb to our attack. The security of Lapin is proven based on the Ring-LPN assumption, and the protocol actually has a relatively similar structure to HB, HB+ and KPCJV. However, Lapin works over a ring R, and the protocol is parametrised by a mapping π from bit-vectors to R. The mapping π is not fully specified in their paper, except that it must have certain properties. It seems somewhat likely that for *arbitrary* π satisfying the properties specified in the Lapin paper, the Lapin protocol may succumb to a similar erasure attack; however, it also seems possible that if π has additional stronger properties, the Lapin protocol might be resilient at least to the specific erasure attacks we have proposed here.

5 Our protocols

In this section, we describe and prove the security of our protocols UbiAuth and UbiAuth+, which achieve concurrent MIM security and ℓ-instance concurrent MIM security (for any *a priori* bounded polynomial ℓ), respectively. The protocols can be instantiated using any secure PRG, and the security proofs are by reduction to PRG security.[12]

The focus of this section is asymptotic analysis and provable security based on abstract cryptographic primitives. We also give some discussion of (theoretical) motivations behind the design of our protocols. In Section 6, we will switch over to a more practical perspective, and consider the performance of UbiAuth instantiated by a concrete stream cipher.

5.1 Ideas and techniques

Notation. For vectors $\mathbf{v}, \mathbf{w} \in \{0,1\}^n$, $\mathbf{v} * \mathbf{w}$ denotes the component-wise (Schur) product, and $\mathbf{v} \cdot \mathbf{w}$ is the field product (in $\mathbb{F}_{2^n}$). For an error-correcting code C, $C.\mathsf{Enc}$ and $C.\mathsf{Dec}$ denote the encoding and decoding functions.

The basic idea of our protocol is that the verifier sends an n-bit random challenge $\mathbf{a}$ to the prover, who responds with an unconditionally secure MAC on $\mathbf{a}$ computed from a secret key that he shares with the verifier.

[12] The standard definitions of pseudorandom generators and pseudorandom function families are given in Appendix A.

The good news is that unconditionally secure MACs (in contrast to computationally secure ones) can give us the efficiency we are after. In particular we will use a MAC of the form $C(\mathbf{a}) * \mathbf{s} \oplus \mathbf{e}$ where $(\mathbf{s}, \mathbf{e})$ is the key, and where C is a linear-time encodable linear code that is good in the sense that both its length and minimum distance are $\Theta(n)$. Then $C(\mathbf{a})$ is the encoding of $\mathbf{a}$ and the product $C(\mathbf{a}) * \mathbf{s}$ is the component-wise (Schur) product. This one-time MAC is linear-time computable and was proven secure in a work of Damgård and Zakarias [12].

However, the bad news is that while $\mathbf{s}$ can be reused over several executions, $\mathbf{e}$ cannot: it must be a fresh random value every time, or the key will be revealed. An obvious solution is to choose $\mathbf{e}$ pseudorandomly using another shared key K. Computing it as $\mathbf{e} = f_K(\mathbf{a})$ where f is PRF may seem natural, but this would be pointless, because then the simpler PRF-based protocol mentioned above might as well be used, and we will again face the efficiency issues associated with using a PRF in this way. To get around this, we propose to have the prover keep a counter i that is incremented for each execution, and compute $\mathbf{e}$ as $\mathbf{e} = f_K(i)$. The point is that now the prover does not need to compute the PRF on arbitrary unpredictable inputs, but only on values that arrive in a particular order $i = 1, 2, \ldots$. To see why this is an advantage, note that the GGM construction forms a tree with exponentially many leaves, one for each possible input. Computing an output requires computing the path to the relevant leaf, calling the PRG once for each vertex. Now the idea is to save the last path we computed and only recompute the part that changes for the next input. If the inputs indeed arrive in order, it turns out that on average we will only need to call the PRG twice per call to the PRF.

We also propose a variant of the GGM tree where every node delivers an output value. This allows us to grow the tree as we go, so that the storage requirement only depends (logarithmically) on how many times *the protocol is actually executed* whereas using the original GGM tree would force us to decide in advance on an upper bound for the number of executions. Goldreich [15] suggested a somewhat related idea where the last leaf of the tree is used as the root of a new one. In the most "extreme" variant of this technique we would get a linear list instead of a tree. This would lead to similar time complexity for the prover, but an adversary could force the verifier to spend much more time than in normal operation, potentially $\Omega(nt)$, where t is the number of times the protocol is executed. The worst case for our approach is $O(n \log t)$.

The construction we just sketched achieves concurrent MIM security: the strongest security notion in the case where the adversary cannot clone or reset the prover. If the adversary can clone or reset, then we need multi-instance concurrent MIM security. We achieve this by modifying our protocol using an additional technique based on universal hashing. In particular, we extend a result of [22] to get a linear time computable ℓ-wise independent hash function family $\mathcal{H}$ for any constant ℓ (rather than pairwise independence as in [22]). If there exists a linear time PRG G that is $(\mathcal{H}, \ell)$-secure, then we get a protocol that is multi-instance concurrent MIM secure and has amortised complexity $O(n)$.

Interestingly, the proof of security is not so straightforward as it may seem at first sight. One naturally expects a simple two-step argument: first use the security of the PRF to replace the pseudorandom $\mathbf{e}$ in the prover's response $C(\mathbf{a}) * \mathbf{s} \oplus \mathbf{e}$ by a truly random value, and then use the unconditional security of the MAC to conclude that the protocol is secure. However, while the first step is fine, the second fails because an adversary against the protocol is more powerful than an adversary against the MAC: he can arbitrarily schedule interactions with the verifier as well as the prover, so we need first to argue that access to the verifier is useless for the adversary. This requires an argument that is completely different from the standard security proof for the MAC.

5.2 Tools for the protocol construction

"One-time" MACs Consider the following simple and unconditionally secure MAC: for a message $\mathbf{a} \in \{0,1\}^n$, the MAC on the message is $\mathbf{a} \cdot \mathbf{s} + \mathbf{e}$, where $(\mathbf{s}, \mathbf{e}) \in \{0,1\}^n \times \{0,1\}^n$ 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 ($\mathbf{s}$) can be reused provided that $\mathbf{e}$ is freshly chosen for each message (see e.g. [3]).

In this work, we focus on a slightly different MAC, which might be considered a variant of the above. For a message $\mathbf{a} \in \{0,1\}^n$, the MAC on the message is $C.\mathsf{Enc}(\mathbf{a}) * \mathbf{s} + \mathbf{e} \in \{0,1\}^{cn}$ where C is an error-correcting code (with constant-fraction distance) with expansion c, and $(\mathbf{s}, \mathbf{e}) \in \{0,1\}^{cn} \times \{0,1\}^{cn}$ is the secret key. The security of this variant MAC is shown in [12].

In our protocols, we consider $\mathbf{s}$ to be the secret key, and generate $\mathbf{e}$ pseudorandomly per execution. 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.

Finally, we remark that although our protocols are presented in terms of the variant MAC, the proofs of correctness and security all go through (with almost no changes) also when using the "$\mathbf{a} \cdot \mathbf{s} + \mathbf{e}$" MAC. Which version is better in a concrete application would be determined by the encoding efficiency of the error-correcting code for the relevant parameters.

Pseudorandom look-up function

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.

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.

Given a PRG G taking an n-bit input, our goal is to generate a series of polynomially many pseudorandom values $r_1, r_2, r_3, \ldots$, such that each r_i can be looked up in time (poly-)logarithmic in i. This is achieved using the tree structure shown in Figure 1.

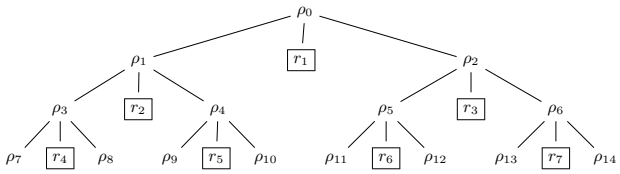

Fig. 1. Pseudorandom lookup tree (first 4 levels)

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 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 3 below. Before proving that the look-up function is a PRF, we give a simple supporting lemma.

Lemma 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*

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

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

Proof. Given in Appendix A.

Theorem 3. *Let G be a PRG and $n, m \in \mathbb{N}$ be positive integers with $m = \mathsf{poly}(n)$. Then the family of functions $\mathcal{F}^{(n,m)} \stackrel{\text{def}}{=} \{\mathsf{lookup}^G_{n,m}(\rho, \cdot)\}_{\rho\in\{0,1\}^n}$ is a PRF with input size n' bits and output size n bits, for any $n' = \mathsf{poly}(n)$.*

Proof. Given in Appendix C.

Looking up random values in order. We would like to look up the random values $\mathsf{lookup}^G_{n,m}(\rho_0, \cdot)$ *in order*, that is, first r_1, then r_2, and so on. This can be done more efficiently than by traversing the tree starting at the root for each new value, essentially by storing the path to the most recently retrieved leaf, and implementing a "next leaf" function which takes the stored path as an input. Naturally, this incurs additional (logarithmic) storage cost, compared to looking up each leaf starting afresh from the root.

In order to apply this method to our lookup tree, we observe that the non-leaf nodes of the lookup tree constitute a binary tree (shown in blue in Figure 2). Applying the path-based lookup algorithm to this binary tree allows in-order retrieval of the first k leaf values in the look-up tree in time $O(k)$, for any $k \in \mathbb{N}$. The storage requirement is $\log(k) \cdot \log(n)$ where n is the size of input to G.

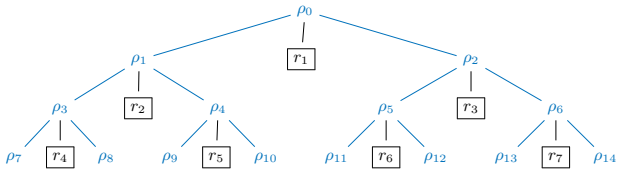

Fig. 2. A binary tree within the lookup tree

Lemma 2. *For any given depth $d \geq 1$, when all the output values (boxed nodes) at depth d of the lookup tree are computed in order (from left to right) by:*

- *first, computing the leftmost output value by traversing the tree downwards from the root,*
- *then, computing each subsequent output value by applying the path-based lookup algorithm to the binary tree of non-leaf nodes up to and including depth $d-1$, and calling the underlying PRG to obtain each actual (leaf) output value,*

the total number C of calls to the underlying PRG G that is required to compute all *the output values at depth d is exactly $2^d + 2^{d-1} - 2$.*

Proof. Given in Appendix D due to space constraints.

Corollary 1. *When computing the values $\mathsf{lookup}^G_{n,m}(\rho_0, i)$ for $i = 1, 2, \ldots$ (in order) by the method described in Lemma 2, the amortised number of calls to G per output value looked up is constant. To be precise, it is less than* 1.5.

Proof. For any given depth $d > 1$, there are 2^{d-1} output values at that depth. By Lemma 2, all of these values can be looked up with a total of $2^d + 2^{d-1} - 2$ calls to G. Hence, the number of calls to G per output value (at depth d) is $\frac{2^d + 2^{d-1} - 2}{2^d} < 1.5$.

5.3 The UbiAuth protocol

Protocol 2. UbiAuth

Public parameters. PRG G, security parameter $n \in \mathbb{Z}$, error-correcting code C with constant-fraction distance and constant expansion factor c.
Key generation. $\mathsf{Ubi.Gen}(1^n)$ samples $\mathbf{s} \leftarrow \{0,1\}^{cn}, \mathbf{s}' \leftarrow \{0,1\}^n$ and outputs secret key $(\mathbf{s}, \mathbf{s}')$.
Initial state. Prover's state consists of $i \in \mathbb{N}$ initialised to 1.

$\mathcal{P}(\mathbf{s}, \mathbf{s}'; i)$		$\mathcal{V}(\mathbf{s}, \mathbf{s}')$
	$\xleftarrow{\mathbf{a}}$	$\mathbf{a} \leftarrow \{0,1\}^n$
$\mathbf{e} := \mathsf{lookup}^G_{n,cn}(\mathbf{s}', i)$ $\mathbf{z} := C.\mathsf{Enc}(\mathbf{a}) * \mathbf{s} + \mathbf{e}$ $i := i + 1$	$\xrightarrow{\mathbf{z}, i}$	accept iff $\mathbf{z} + C.\mathsf{Enc}(\mathbf{a}) * \mathbf{s} =$ $\mathsf{lookup}^G_{n,cn}(\mathbf{s}', i)$

The magenta color in the protocol indicates (updating of) the prover's state.

It is clear that UbiAuth is perfectly complete, since $\mathsf{lookup}^G_{n,cn}$ is deterministic. We next prove that Protocol 2 is actively secure (Lemma 3) which serves as a stepping-stone to proving concurrent MIM security (Theorem 4).

Lemma 3. UbiAuth *is secure against active attacks.*

Proof. Let e_j denote the noise string for index j. 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 $\mathcal{A}$ play the active security game as before, except that $\mathcal{P}, \mathcal{V}$ no longer know $\mathbf{s}'$, but instead have oracle access to $\mathsf{lookup}^G_{n,n}(\mathbf{s}', \cdot)$.

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

Games 1 and 2 are perfectly indistinguishable for the adversary, since the messages sent by are distributed identically in the two games. 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) $\mathsf{lookup}^G_{n,n}(\mathbf{s}', \cdot)$ and a random oracle – this contradicts Theorem 3. Therefore, Games 1, 2, and 3 are computationally indistinguishable, and so the e_j are indistinguishable from uniformly random noise.

We have established that the prover's message $z = C.\mathsf{Enc}(a) * s + e_j$ is indistinguishable from $C.\mathsf{Enc}(a) * s + r$ for random r. Hence, z is indistinguishable from random to any active adversary, regardless of the choice of a. It remains only to consider the interaction of $\mathcal{A}$ with the honest verifier $\mathcal{V}$. Given a challenge a from $\mathcal{V}$, $\mathcal{A}$ can have at most negligible advantage at guessing the (unique) value of z that $\mathcal{V}$ will accept, as shown 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 $C.\mathsf{Enc}(a) * s$, 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 $z - z' = (C.\mathsf{Enc}(a) - C.\mathsf{Enc}(a')) * s$. This happens with negligible probability since all of a, a', z, z' were chosen independently of s.

Theorem 4. UbiAuth *is secure against concurrent MIM attacks.*

Proof. We show that if there is an adversary $\mathcal{A}$ which achieves a certain advantage when conducting a concurrent MIM attack, then there is another adversary $\mathcal{A}'$ that *only talks to the prover* in Protocol 2 and achieves essentially the same advantage. First, we 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 concurrent MIM attack, there is an equally successful *active* attack, and finally refer to Lemma 3 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, there are two cases to consider:

1. $\mathcal{A}$ previously received answer z', j from $\mathcal{P}$, where $z' = C.\mathsf{Enc}(a') * s + e_j$ 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. $\mathcal{V}$'s challenge is distributed identically to that of $\mathcal{V}'$. In case 1a, $\mathcal{V}$ will accept if and only if z has the correct value $C.\mathsf{Enc}(a) * s + e_j$: so $\mathcal{V}'$ always makes the same decision as $\mathcal{V}$. In case 1b, V only accepts if $z = C.\mathsf{Enc}(a) * s + e_j$, but since $z' = C.\mathsf{Enc}(a') * s + e_j$ it must be that $(z - z') = (C.\mathsf{Enc}(a) - C.\mathsf{Enc}(a')) * s$. This happens with negligible probability because s is random and z, z', a, a' are all independent of s: $\mathcal{P}$'s responses, including z', are independent of s; and since this is the first query, $\mathcal{V}$ has not seen s yet, so a, a' and z must be independent of s too. Thus, $\mathcal{V}$ rejects with overwhelming probability, so $\mathcal{V}'$ is statistically close to the right behavior. Finally, in case 2, no one sees e_j before $\mathcal{A}$ produces z, j. If $\mathcal{V}$ accepts, we have $z = C.\mathsf{Enc}(a) * s + e_j$, so $e_j = z - C.\mathsf{Enc}(a) * s$, which happens with negligible probability since z, a and s are independent of e_j.

Therefore, we can replace $\mathcal{V}$ with $\mathcal{V}'$ for the first query, and $\mathcal{A}$'s advantage changes at most negligibly as a result. Repeating this argument for all the queries, we reach the game where $\mathcal{V}$ is entirely replaced by $\mathcal{V}'$, and $\mathcal{A}$'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 $\mathcal{V}$). The result then follows from Lemma 3.

Linear-time implementation. The prover in Protocol 2 can run in time $O(n)$ and space $O(\log(n) \cdot \log(k))$, where k is the number of protocol executions run so far[13]: this is possible by using the path-based lookup algorithm to compute $\mathsf{lookup}^G_{n,cn}(\cdot,\cdot)$ as described in Lemma 2. This follows from Corollary 1, and the fact that there exist linear-time, linear-stretch PRGs and linear-time encodable codes with constant-factor expansion and large constant-fraction distance (such as those of Guruswami and Indyk [19]).

The verifier can also be implemented to run in linear time *for honest executions*, by using the same method as the prover to compute $\mathsf{lookup}^G_{n,cn}(\cdot,\cdot)$. Clearly, if the prover is honest, the verifier will run in linear time. If the prover cheats and breaks the sequence, then the verifier can retrieve the required $\mathsf{lookup}^G_{n,cn}(\cdot,\cdot)$ value by the "backup method" of traversing the lookup tree downwards from the root, which takes $O(n \cdot \log(k))$ time instead. This implementation requires $N \cdot O(\log(n) \cdot \log(k))$ space, where N is the number of different provers with which the verifier interacts. Note that since the multiplication can be done in depth $O(\log(n))$, if the PRG G is of poly-logarithmic depth, then the verifier does only poly-logarithmic depth computation (even when the prover cheats).

5.4 Security against memory erasures

As presented in Section 5.3, UbiAuth is *not* necessarily secure against memory erasure attacks, depending on the underlying PRG. In Appendix **??**, we give an example of a PRG G such that if UbiAuth were instantiated using G, the resulting protocol would easily succumb to erasure attacks.

In this subsection, we describe a "reinforced" variant of UbiAuth, called r-UbiAuth, which achieves security against concurrent MIM attacks with erasures. In a nutshell, our method takes any PRG G and constructs a *reinforced lookup function* r-lookup based on G, which maintains unpredictability even when adversary can obtain outputs from a "partially erased" seed. Of course, unpredictability is only maintained up to a point: if the adversary erases the entire seed, then we clearly cannot hope for any unpredictability. We achieve this strong unpredictability property at the cost of a factor-n^2 increase in key size.[14]

[13] In other words, k is the number of leaf values in the lookup tree that have been retrieved so far. Note that if desired, the value of k can be upper-bounded by some polynomial-size K, by "starting a new tree" after K values have been retrieved from the initial tree: the $(K+1)^{th}$ leaf value in the first tree serves as the root of a new tree in which subsequent lookups are done. This technique was suggested in [15].

[14] Actually, the construction could be made to work with a factor-n^ϵ increase in key size for any $\epsilon > 1$.

We then modify the protocol description, so that seeds with "too many zeros" are ignored: that is, the prover will refuse to output answers if its seed contains too many zeros. Informally, our proof of security argues that either the adversary has performed few enough erasures that unpredictability still holds, or the prover will have stopped answering and so the adversary cannot obtain further information.

Definition 10 (Reinforced lookup function). *Let G be a PRG. Define $\mathfrak{R}_n$ to be the set of n^3-bit strings of Hamming weight between $\lfloor (n^3 - n^2)/2 \rfloor$ and $\lfloor (n^3 + n^2)/2 \rfloor$ (inclusive). We define the* reinforced lookup function $\mathsf{r\text{-}lookup}$*, which takes as input a n^3-bit seed $\rho \in \{0,1\}^{n^3}$.*

$$\mathsf{r\text{-}lookup}^G_{n,m}((\rho_1, \ldots, \rho_{n^2}), i) \stackrel{\text{def}}{=} \begin{cases} \mathsf{lookup}^G_{n,m}(\rho_w, i) & \text{if } \rho \in \mathfrak{R}_n \\ 0^m & \text{otherwise} \end{cases},$$

where $\rho_i \in \{0,1\}^n$ for each $j \in [n^2]$, and the n^3-bit seed is $\rho = (\rho_1, \ldots, \rho_{n^2})$ and $w = w(\rho) \stackrel{\text{def}}{=} ||\rho||_1 \bmod n^2 + 1$ is the Hamming weight of the seed.

That the reinforced lookup function is a PRF (Theorem 5, below) can be reduced to the fact that the original lookup function is a PRF (Theorem 3).

Theorem 5. *Let G be a PRG and $n, m \in \mathbb{N}$ be positive integers with $m = \mathsf{poly}(n)$. Let $\mathfrak{S}_n$ denote the set of all n-bit strings of weight between $\lfloor (n - \sqrt{n})/2 \rfloor$ and $\lceil (n + \sqrt{n})/2 \rceil$ (inclusive). Then for any $n' = \mathsf{poly}(n)$, the family of functions*

$$\mathcal{F}'^{(n,m)} \stackrel{\text{def}}{=} \{\mathsf{r\text{-}lookup}^G_{n,m}(\rho, \cdot)\}_{\rho \in \mathfrak{R}_n}$$

is a PRF with input size n' bits and output size n bits, where the seed $\rho = (\rho_1, \ldots, \rho_n)$ is sampled from the product distribution $\mathfrak{D}^n$ where $\mathfrak{D}$ is the uniform distribution over $\mathfrak{S}_n$.

Proof. By a Chernoff bound, $\Pr_{\rho \leftarrow \mathfrak{D}^n}[\rho \notin \mathfrak{R}_n]$ is negligible. We therefore disregard this case (in which the function selected by the seed is the zero function).

Let $\mathcal{F}^{(n,m)} \stackrel{\text{def}}{=} \{\mathsf{lookup}^G_{n,m}(\rho, \cdot)\}_{\rho \in \{0,1\}^n}$, i.e., the PRF defined by the original lookup function. By definition of $\mathsf{r\text{-}lookup}$, for any n^3-bit seed $\rho' \in \mathfrak{R}_n$ of $\mathcal{F}'^{(n,m)}$, there is an *induced* n-bit seed $\rho \in \{0,1\}^n$, which is a substring of ρ', such that $\mathsf{r\text{-}lookup}^G_{n,m}(\rho', \cdot) = \mathsf{lookup}^G_{n,m}(\rho, \cdot)$. Let $\mathfrak{D}'$ denote the distribution on n-bit seeds which is thus *induced* by n^3-bit seeds sampled according to $\mathfrak{D}^n$. The support of $\mathfrak{D}'$ is the set $\mathfrak{S}_n$ (defined in the theorem statement). By a Chernoff bound, $\mathfrak{S}_n$ has constant probability mass in the uniform distribution on n bits (i.e., the distribution of seeds for $\mathcal{F}^{(n,m)}$). It follows that if there is an algorithm $\mathcal{A}'$ that distinguishes with non-negligible advantage ε between oracle access to a random function in $\mathcal{F}'^{(n,m)}$ and a truly random m-bit function, then $\mathcal{A}'$ will also distinguish with non-negligible advantage $O(\varepsilon)$ between oracle access to a random function in $\mathcal{F}^{(n,m)}$ and a truly random function. This contradicts Theorem 3 (i.e., that $\mathcal{F}^{(n,m)}$ is a PRF).

Now we define the protocol r-UbiAuth which makes use of the reinforced lookup function. It is the same as UbiAuth, except that the reinforced lookup function is used in place of the original lookup function, and if the prover finds itself with an invalid seed (i.e., one which is not in $\mathfrak{R}_n$), it stops answering.

Protocol 3. r-UbiAuth

Public parameters. PRG G, security parameter $n \in \mathbb{Z}$, error-correcting code C with constant-fraction distance and constant expansion factor c.
Key generation. r-Ubi.Gen(1^n) samples $\mathbf{s} \leftarrow \{0,1\}^{cn}, \mathbf{s}' \leftarrow \mathfrak{R}_n \subset \{0,1\}^{n^3}$ and outputs secret key $(\mathbf{s}, \mathbf{s}')$.
Initial state. Prover's state consists of $i \in \mathbb{N}$ initialised to 1.

$\mathcal{P}(\mathbf{s}, \mathbf{s}'; i)$		$\mathcal{V}(\mathbf{s}, \mathbf{s}')$
	$\xleftarrow{\mathbf{a}}$	$\mathbf{a} \leftarrow \{0,1\}^n$
if $\mathbf{s}' \notin \mathfrak{R}_n$, abort $\mathbf{e} := \mathsf{r\text{-}lookup}^G_{n,cn}(\mathbf{s}', i)$ $\mathbf{z} := C.\mathsf{Enc}(\mathbf{a}) * \mathbf{s} + \mathbf{e}$ $i := i + 1$	$\xrightarrow{\mathbf{z}, i}$	accept iff $\mathbf{z} + C.\mathsf{Enc}(\mathbf{a}) * \mathbf{s} =$ $\mathsf{r\text{-}lookup}^G_{n,cn}(\mathbf{s}', i)$

A theorem and detailed proof sketch for the security of r-UbiAuth against concurrent MIM attacks with erasures is given in Appendix E.

5.5 The UbiAuth+ protocol

The UbiAuth+ protocol, shown in Protocol 4, achieves ℓ-instance concurrent MIM security for any polynomial ℓ. Moreover, if ℓ is constant, we can still get (amortised) linear time. UbiAuth+ makes use of ℓ-wise independent hashing, which is defined below.

Definition 11. *A function family $\mathcal{H}$ of functions that map n bits to m bits is a **ℓ-wise independent hash function family** if for all $y_1, \ldots, y_\ell \in \{0,1\}^m$ and for all distinct $x_1, \ldots, x_\ell \in \{0,1\}^n$, it holds that*

$$\Pr_{h \leftarrow \mathcal{H}}[h(x_1) = y_1 \wedge h(x_2) = y_2 \wedge \cdots \wedge h(x_\ell) = y_\ell] = 2^{-\ell m} \ .$$

Our security proofs follow a similar structure to those of Protocol 2: we first prove ℓ-instance active security, then use this to prove ℓ-instance concurrent MIM security. The proofs are deferred to Appendix D, due to space constraints.

Lemma 4. UbiAuth+ *is secure against ℓ-instance active attacks.*

Theorem 6. UbiAuth+ *is secure against ℓ-instance concurrent MIM attacks.*

Protocol 4. UbiAuth+

Public parameters. PRG G, security parameter $n \in \mathbb{Z}$, error-correcting code C with constant-fraction distance and constant expansion factor cm function family $\mathcal{H} = \{h_{\mathbf{r}}\}_{\mathbf{r}\in\{0,1\}^{\beta}}$ of 2ℓ-wise independent hash functions mapping $(\ell+1)\cdot n$ bits to n bits.
Key generation. $\mathsf{Ubi}^{+}.\mathsf{Gen}(1^n)$ samples $\mathbf{s} \leftarrow \{0,1\}^{cn}, \mathbf{s}' \leftarrow \{0,1\}^n$ and outputs secret key $(\mathbf{s}, \mathbf{s}')$.
Initial state. Prover's state consists of $i \in \mathbb{N}$ initialised to 1.

$\mathcal{P}(\mathbf{s}, \mathbf{s}'; i)$		$\mathcal{V}(\mathbf{s}, \mathbf{s}')$
	$\xleftarrow{\mathbf{a}}$	$\mathbf{a} \leftarrow \{0,1\}^n$
$\mathbf{r}_i := \mathsf{lookup}^G_{n,(\ell+1)\cdot n+\beta}(\mathbf{s}', i)$		
$\mathbf{e} := h_{\mathbf{r}_i^{[1,\beta]}}\left(\mathbf{a} + \mathbf{r}_i^{[\beta+1,(\ell+1)\cdot n+\beta]}\right)$		$\mathbf{r}_i := \mathsf{lookup}^G_{n,(\ell+1)\cdot n+\beta}(\mathbf{s}', i)$
$\mathbf{z} := C.\mathsf{Enc}(\mathbf{a}) * \mathbf{s} + \mathbf{e}$	$\xrightarrow{\mathbf{z}, i}$	accept iff $\mathbf{z} + C.\mathsf{Enc}(\mathbf{a}) * \mathbf{s} = h_{\mathbf{r}_i^{[1,\beta]}}\left(\mathbf{a} + \mathbf{r}_i^{[\beta+1,(\ell+1)\cdot n+\beta]}\right)$
$i := i + 1$		

Security against erasures. A variant of UbiAuth+ can moreover be secure against concurrent MIM attacks with erasures, using a construction based on "reinforced" PRGs which is exactly analogous to that described in Section 5.4.

Linear-time implementation. If ℓ is constant, then the prover in Protocol 4 can run in (amortised) time $O(n)$ and space $O(\log(n) \cdot \log(k))$, where k is the number of protocol executions run so far: as with Protocol 2, this requires the use of the path-based lookup algorithm to compute $\mathsf{lookup}^G_{n,cn}(\cdot,\cdot)$, and the use of a linear-time PRG and linear-time encodable code. In addition, we require an ℓ-wise independent hash function family whose functions can be sampled and computed in linear time. A construction of a hash function family satisfying these properties for constant ℓ is given in Appendix F. As in the case of Protocol 2, the verifier in Protocol 4 can also be implemented to run in linear time when the prover is honest.

6 Instantiating with Grain-128

Returning to a more practical perspective, we consider how UbiAuth would perform when instantiated with a stream cipher acting as the PRG. Given the motivation of lightweight hardware devices, we focus on a hardware-oriented cipher and compare performance at the relatively low 100kHz clock frequency which is common in low-end devices. By the same reasoning, we analyse the *prover's* performance which is the bottleneck in these applications.

Our analysis considers our protocol instantiated with the Grain-128 stream cipher as the PRG. We selected Grain-128 from the eSTREAM (Profile 2)[15] portfolio because it is optimised for hardware implementation with high throughput, and it has a mode with 128 bits of security which is good for comparison to AES with 128-bit keys. Moreover, Grain has the advantage of a relatively fast initialization time compared to other Profile 2 candidates such as Trivium [14]. Specifically, our benchmarks are based on the performance of Grain128x32 and a comparable hardware implementation of AES (by [35]), as documented in [17, Table 4].

Simple PRF-based protocol		**Our protocol**	
AES	Grain (+GGM)	Grain (32-bit state)	Grain (2048-bit state)
0.54	10.24	3.84	0.18

Table 1. Amortised prover *computation time* (in ms) in two-round authentication

An interesting initial observation is that if one uses Grain128x32 in the GGM construction of a PRF, then the speed of a Grain-based PRF evaluation is "only" about 20 times slower than an AES evaluation, as shown in the second column of Table 1. We have included the second column to highlight that the efficiency advantage of stream ciphers over block ciphers is a major factor that counterbalances the relative inefficiency of the GGM-style construction that is inherent in our protocols, and not to suggest that GGM might be a reasonable solution in practice!

Indeed, as shown in Table 1, our protocol outperforms the simple AES-based protocol) *in terms of prover computation time* by a factor of three when instantiated with 2048 bits of prover state. Our figures do not account for write time to update state, in order to have a fairer comparison of the computation times, since write times vary greatly across different chips / types of memory.[16]

Size of the prover's state. In Table 1, "32-bit state" refers to the scenario when the prover stores only a (32-bit) counter value, whereas "4096-bit state" refers to the variant when the prover stores partial paths in the lookup tree.

Remarks on protocol implementation. Note that in software, using a more standard MAC of form $a \cdot s + e$, where the product is in the field with 2^n elements, may be an advantage (even if it is not preferable asymptotically). For instance, for $n = 64$, we can multiply by a fixed element s using 8 table look-ups in 16 KB of precomputed tables. However, if the prover is a small hardware device, whereas the verifier has more power, then using the MAC $C(a) * s + e$ that we

[15] Profile 2 ciphers are described as "Stream ciphers for hardware applications with restricted resources such as limited storage, gate count, or power consumption."

[16] To give a very rough estimate: based on the specifications of common RFID manufacturers, the write time to non-volatile memory is likely to range between less than millisecond to a few milliseconds, but varies depending on the exact hardware.

suggest may be even better, since we can ask the verifier to send the encoding $C(a)$ rather than a. Our proof of security still applies if the prover checks that he receives a codeword, and this can be extremely fast in hardware if we use an LDPC (low-density parity check) code: the codeword check consists of computing parity functions of a small constant (say, 3 or 4) number of bits, and all the parity functions could be computed in parallel. This reduces the prover's computation essentially to the cost of running the stream cipher (and we have assumed that this is the case, in computing the figures in Table 1).

Remark on counter mode. The use of a counter in our protocol may prompt the question: if we use a stream cipher with a nonce in counter mode, then could we run the stream cipher on consecutive counter values instead of evaluating the lookup function? The answer is that such a scheme would only be secure if the stream cipher were *secure against adversarially chosen nonces*. The standard assumption about stream ciphers is that nonces need not be kept secret, but are chosen honestly (rather than adversarially) by the party in possession of the key [36]. The use of the lookup function allows us to base our protocol on *any* PRG, or *any* stream cipher. The behaviour of stream ciphers under adversarially chosen nonces seems to be little studied/understood: for example, the eSTREAM cipher candidates do not discuss the possibility of adversarial nonces. Assuming security under adversarial nonces is equivalent to assuming "PRF security" of a stream cipher, rather than "PRG security". The only instance of such an assumption being used in the literature, of which we are aware, is [4] which proposes a (three-round) authentication protocol based on "PRF security of a stream cipher", but gives no discussion of why such an assumption may be reasonable. Indeed, the literature on the effects of *nonce misuse* in the context of authenticated encryption (a line of work started by [34]) would seem to suggest that one should be very wary of this sort of assumption.

7 Acknowledgements

We would like to thank Ron Rivest, Srini Devadas, and Chiraag Juvekar for illuminating conversations about lightweight authentication. We thank an anonymous reviewer for bringing to our attention the topic of stream ciphers in counter mode. Sunoo Park acknowledges support from the MACS project NSF grant CNS-1413920 and a Simons Investigator Award Agreement dated 2012-06-05.

References

1. B. Applebaum. Pseudorandom generators with long stretch and low locality from random local one-way functions. *SIAM Journal on Computing*, 42(5):2008–2037, 2013.
2. M. Bellare and P. Rogaway. Entity authentication and key distribution. In D. R. Stinson, editor, *Advances in Cryptology - CRYPTO '93, 13th Annual International Cryptology Conference, Santa Barbara, California, USA, August 22-26, 1993, Proceedings*, volume 773 of *Lecture Notes in Computer Science*, pages 232–249. Springer, 1993.

3. R. Bendlin, I. Damgård, C. Orlandi, and S. Zakarias. Semi-homomorphic encryption and multiparty computation. In Paterson [32], pages 169–188.
4. O. Billet, J. Etrog, and H. Gilbert. Lightweight privacy preserving authentication for RFID using a stream cipher. In S. Hong and T. Iwata, editors, *Fast Software Encryption, 17th International Workshop, FSE 2010, Seoul, Korea, February 7-10, 2010, Revised Selected Papers*, volume 6147 of *Lecture Notes in Computer Science*, pages 55–74. Springer, 2010.
5. S. Bono, M. Green, A. Stubblefield, A. Juels, A. D. Rubin, and M. Szydlo. Security analysis of a cryptographically-enabled RFID device. In P. McDaniel, editor, *Proceedings of the 14th USENIX Security Symposium, Baltimore, MD, USA, July 31 - August 5, 2005*. USENIX Association, 2005.
6. F. Brown. RFID hacking: Live free or die hard, 2013. Presented at Black Hat USA 2013: `https://www.blackhat.com/us-13/briefings.html#Brown`.
7. R. Canetti, Y. Dodis, S. Halevi, E. Kushilevitz, and A. Sahai. Exposure-resilient functions and all-or-nothing transforms. In B. Preneel, editor, *Advances in Cryptology - EUROCRYPT 2000, International Conference on the Theory and Application of Cryptographic Techniques, Bruges, Belgium, May 14-18, 2000, Proceeding*, volume 1807 of *Lecture Notes in Computer Science*, pages 453–469. Springer, 2000.
8. D. Cash, E. Kiltz, and S. Tessaro. Two-round man-in-the-middle security from LPN. In *TCC*, 2016.
9. B. Chor, O. Goldreich, J. Håstad, J. Friedman, S. Rudich, and R. Smolensky. The bit extraction problem of t-resilient functions (preliminary version). In *26th Annual Symposium on Foundations of Computer Science, Portland, Oregon, USA, 21-23 October 1985*, pages 396–407. IEEE Computer Society, 1985.
10. N. Courtois, K. Nohl, and S. O'Neil. Algebraic attacks on the crypto-1 stream cipher in mifare classic and oyster cards. *IACR Cryptology ePrint Archive*, 2008:166, 2008.
11. I. Damgård, S. Faust, P. Mukherjee, and D. Venturi. Tamper resilient cryptography without self-destruct. *IACR Cryptology ePrint Archive*, 2013:124, 2013.
12. I. Damgård and S. Zakarias. Constant-overhead secure computation of boolean circuits using preprocessing. In *TCC*, pages 621–641, 2013.
13. Y. Dodis, E. Kiltz, K. Pietrzak, and D. Wichs. Message authentication, revisited. In D. Pointcheval and T. Johansson, editors, *EUROCRYPT*, volume 7237 of *Lecture Notes in Computer Science*, pages 355–374. Springer, 2012.
14. M. Feldhofer and J. Wolkerstorfer. *Hardware Implementation of Symmetric Algorithms for RFID Security*, pages 373–415. Springer, 2008.
15. O. Goldreich. *Foundations of Cryptography: Basic Techniques*. Cambridge University Press, June 2001.
16. O. Goldreich, S. Goldwasser, and S. Micali. How to construct random functions. *J. ACM*, 33(4):792–807, 1986.
17. T. Good and M. Benaissa. Hardware performance of estream phase-iii stream cipher candidates. In *In SASC*, 2008.
18. A. Greenberg. Hackers remotely kill a jeep on the highway – with me in it, July 2015. `https://www.wired.com/2015/07/hackers-remotely-kill-jeep-highway`.
19. V. Guruswami and P. Indyk. Linear-time encodable/decodable codes with near-optimal rate. *IEEE Transactions on Information Theory*, 51(10):3393–3400, 2005.
20. S. Heyse, E. Kiltz, V. Lyubashevsky, C. Paar, and K. Pietrzak. Lapin: An efficient authentication protocol based on ring-LPN. In A. Canteaut, editor, *FSE*, volume 7549 of *Lecture Notes in Computer Science*, pages 346–365. Springer, 2012.

21. N. J. Hopper and M. Blum. Secure human identification protocols. In C. Boyd, editor, *Advances in Cryptology - ASIACRYPT 2001, 7th International Conference on the Theory and Application of Cryptology and Information Security, Gold Coast, Australia, December 9-13, 2001, Proceedings*, volume 2248 of *Lecture Notes in Computer Science*, pages 52–66. Springer, 2001.
22. Y. Ishai, E. Kushilevitz, R. Ostrovsky, and A. Sahai. Cryptography with constant computational overhead. In C. Dwork, editor, *Proceedings of the 40th Annual ACM Symposium on Theory of Computing, Victoria, British Columbia, Canada, May 17-20, 2008*, pages 433–442. ACM, 2008.
23. A. Juels and S. A. Weis. Authenticating pervasive devices with human protocols. In V. Shoup, editor, *CRYPTO*, volume 3621 of *Lecture Notes in Computer Science*, pages 293–308. Springer, 2005.
24. J. Katz, J. S. Shin, and A. Smith. Parallel and concurrent security of the HB and HB^+ protocols. *J. Cryptology*, 23(3):402–421, 2010.
25. E. Kiltz, K. Pietrzak, D. Cash, A. Jain, and D. Venturi. Efficient authentication from hard learning problems. In Paterson [32], pages 7–26.
26. K. Koscher, A. Czeskis, F. Roesner, S. Patel, T. Kohno, S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham, and S. Savage. Experimental security analysis of a modern automobile. In *31st IEEE Symposium on Security and Privacy, S&P 2010, 16-19 May 2010, Berkeley/Oakland, California, USA*, pages 447–462. IEEE Computer Society, 2010.
27. J. Lee and Y. Yeom. Efficient rfid authentication protocols based on pseudorandom sequence generators. *Designs, Codes and Cryptography*, 51(2):195–210, 2008.
28. V. Lyubashevsky and D. Masny. Man-in-the-middle secure authentication schemes from LPN and weak prfs. In R. Canetti and J. A. Garay, editors, *Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part II*, volume 8043 of *Lecture Notes in Computer Science*, pages 308–325. Springer, 2013.
29. Y. Mansour, N. Nisan, and P. Tiwari. The computational complexity of universal hashing. In *Proceedings of the Twenty-second Annual ACM Symposium on Theory of Computing*, STOC '90, pages 235–243, 1990.
30. E. J. Markey and R. Blumenthal. Security and privacy in your car act of 2015 (SPY car act), 2015. `http://www.markey.senate.gov/imo/media/doc/SPY%20Car%20legislation.pdf`.
31. C. Miller and C. Valasek. Remote exploitation of an unaltered passenger vehicle, 2015. Presented at Black Hat USA 2015: `https://www.blackhat.com/us-15/briefings.html#chris-valasek`.
32. K. G. Paterson, editor. *Advances in Cryptology - EUROCRYPT 2011 - 30th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tallinn, Estonia, May 15-19, 2011. Proceedings*, volume 6632 of *Lecture Notes in Computer Science*. Springer, 2011.
33. D. Pointcheval and G. Poupard. A new np-complete problem and public-key identification. *Des. Codes Cryptography*, 28(1):5–31, 2003.
34. P. Rogaway and T. Shrimpton. A provable-security treatment of the key-wrap problem. In S. Vaudenay, editor, *Advances in Cryptology - EUROCRYPT 2006, 25th Annual International Conference on the Theory and Applications of Cryptographic Techniques, St. Petersburg, Russia, May 28 - June 1, 2006, Proceedings*, volume 4004 of *Lecture Notes in Computer Science*, pages 373–390. Springer, 2006.
35. A. Satoh, S. Morioka, K. Takano, and S. Munetoh. A compact rijndael hardware architecture with s-box optimization. In C. Boyd, editor, *Advances in Cryptology —*

ASIACRYPT 2001: 7th International Conference on the Theory and Application of Cryptology and Information Security Gold Coast, Australia, December 9–13, 2001 Proceedings. Springer Berlin Heidelberg, 2001.
36. B. Schneier. *Applied Cryptography (2Nd Ed.): Protocols, Algorithms, and Source Code in C*. John Wiley & Sons, Inc., New York, NY, USA, 1995.
37. S. P. Skorobogatov and R. J. Anderson. Optical fault induction attacks. In B. S. K. Jr., Ç. K. Koç, and C. Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2002, 4th International Workshop, Redwood Shores, CA, USA, August 13-15, 2002, Revised Papers*, volume 2523 of *Lecture Notes in Computer Science*, pages 2–12. Springer, 2002.
38. S. Vadhan and C. J. Zheng. Characterizing pseudoentropy and simplifying pseudorandom generator constructions. In *Proceedings of the 44th symposium on Theory of Computing*, pages 817–836. ACM, 2012.
39. T. Zillner. Zigbee exploited: The good, the bad and the ugly, 2015. Presented at Black Hat USA 2015: `https://www.blackhat.com/us-15/briefings.html#zigbee-exploited-the-good-the-bad-and-the-ugly`. Whitepaper available online.

A Pseudorandom primitives

In this section we give the standard definitions of pseudorandom generators (PRGs) and pseudorandom function families (PRFs), and prove a small lemma.

Definition 12 (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 13 (Pseudorandom function family (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.*

Lemma 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*

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

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

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

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

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 $|p_0-p_q|\geq P(n)$. Then, since $p_0-p_q=\sum_{i\in[q]}(p_{i-1}-p_i)$, there must exist $i^*\in[q]$ 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 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 cases. This contradicts that G is a PRG.

B 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 3. $\widetilde{H}_{k^*}$ 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

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

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.

C Ordered traversal of leaves of a binary tree

Theorem 3 *Let G be a PRG and $n, m \in \mathbb{N}$ be positive integers with $m = \mathsf{poly}(n)$. Then the family of functions $\mathcal{F}^{(n,m)} \stackrel{\text{def}}{=} \{\mathsf{lookup}^G_{n,m}(\rho, \cdot)\}_{\rho \in \{0,1\}^n}$ is a PRF with input size n' bits and output size n bits, for any $n' = \mathsf{poly}(n)$.*

Proof.

Algorithm 2. Binary tree traversal

```
1  Path pathToNextLeaf(int depth,
2                      Path currentPath, int currentLeafNum) {
3    if (depth = 1) {
4      Leaf nextLeaf = currentPath.root.rightChild();
5      return currentPath.removeEndNode().append(nextLeaf);
6    } else if (depth > 1) {
7      if (currentLeafNum is even) {
8        Leaf nextLeaf = currentPath.endNode.rightChild();
9        return currentPath.removeEndNode().append(nextLeaf);
10     } else {
11       Path pathToParent = currentPath.removeEndNode();
12       int parentLeafNum = floor(currentLeafNum/2);
13       Path pathToNextParent =
14         pathToNextLeaf(depth-1, pathToParent, parentLeafNum);
15       return pathToNextParent.append(
16         pathToNextParent.endNode.leftChild());
17     }
18   }
19 }
```

Lemma 5. *For any given depth d, when Algorithm 2 is used to compute all the leaves of a complete binary tree (of depth d) in order, the* `leftChild()` *and* `rightChild()` *methods are called exactly once for each non-leaf node in the*

tree. More precisely, the method employed is the following: in order to obtain the first leaf, the standard method traversing the tree downwards from the root is used; and then subsequent leaves are obtained in order by calling `pathToLeaf1 = nextLeafPath(d, pathToLeaf0, 0)`*, then* `pathToLeaf2 = nextLeafPath(d, pathToLeaf1, 1)`*, etc.*

Proof. In order to obtain the the first leaf node (and the path thereto), we need to call `leftChild()` exactly once on all nodes along that path. When $d = 1$, this means that `leftChild()` is called on the root node when obtaining the first leaf node. Moreover, when $d = 1$, it is clear (from lines 2-4) that `rightChild()` is called on the root node exactly once (when obtaining the second leaf node). Hence, the lemma holds for $d = 1$.

For $d > 1$, we argue by induction. It is sufficient to prove that for any $d > 1$:

1. `pathToNextLeaf` is called[17] exactly once for every node at depth $d-1$, except the final node at depth $d-1$ (since for that node, there is no next leaf); and
2. `leftChild()` and `rightChild()` are called exactly once on each node at depth $d-1$.

We call `nextLeafPath()` once for each `currentLeafNum` $\in \{0, \ldots, 2^{\texttt{depth}} - 2\}$ (from the statement of the lemma). In particular, `nextLeafPath()` is called once for the left child of each node at depth `depth`-1 (these nodes are exactly those for which `currentLeafNum` is even). From lines 6-8, it follows that for every node at depth `depth`-1, the `rightChild()` method is called exactly once at line 7.

The recursive call to `pathToNextLeaf()` occurs on line 12, and is only executed when `currentLeafNum` is odd (by the if-clause on lines 6-14). To be precise, the odd values of `currentLeafNum` for which we run `pathToNextLeaf()` are $\{1, 3, \ldots, 2^{\texttt{depth}} - 3\}$. This corresponds exactly to the set of right-children of nodes at depth `depth`-1, except the last one. We see on line 12 that the path passed into the recursive call is `pathToParent`, the path to the parent of the current node. Thus, `pathToNextLeaf()` is recursively called exactly once for each node at depth `depth`-1, except the last one. This satisfies condition 1.

Finally, for each path (of depth `depth`-1) which is *returned* by a recursive call to `pathToNextLeaf()`, `leftChild()` is called on the end node of the path (on line 13). Since we have already established that `pathToNextLeaf()` is recursively called exactly once for each node at depth `depth`-1, except the last node, and the functionality of `pathToNextLeaf()` is to return the next node at a given depth, it follows that the paths returned by such recursive calls to `pathToNextLeaf()` lead to each node at depth `depth`-1, except the *first* node. We conclude that `leftChild()` is called exactly once for each node at depth `depth`-1, except the first node. Moreover, `leftChild()` is called on the first node at depth `depth`-1 when we initially retrieve the first leaf. So, `leftChild()` is called exactly once on each node at depth `depth`-1. In conjunction with our earlier observatiosn

[17] To be precise, when we write "`pathToNextLeaf` is called for a given node" we mean that `pathToNextLeaf(depth, path, leafNum)` is called, where `depth` is the depth of the node in the tree, `path` is the path from the root to that node, and `leafNum` is the number of the node when counting the nodes at depth `depth` from left to right.

about calls to `rightChild()`, this means that condition 2 is satisfied. The result follows.

D Security proofs

Lemma 2 *For any given depth $d \geq 1$, when all the output values (boxed nodes) at depth d of the lookup tree are computed in order (from left to right) by:*

- *first, computing the leftmost output value by traversing the tree downwards from the root,*
- *then, computing each subsequent output value by applying the path-based lookup algorithm to the binary tree of non-leaf nodes up to and including depth $d-1$, and calling the underlying PRG to obtain each actual (leaf) output value,*

the total number C of calls to the underlying PRG G that is required to compute all *the output values at depth d is exactly $2^d + 2^{d-1} - 2$.*

Proof. By the construction of the lookup tree, `leftChild()` and `rightChild()` can each be implemented by a single invocation of the PRG G. When the path-based lookup algorithm is used to look up all nodes at a particular depth $d-1$ in a binary tree (in order), the `leftChild()` and `rightChild()` methods will each be called exactly once for every node at depth less than $d-1$ in the tree. Thus, G will be called twice for every node at depth less than $d-1$ in the binary tree. There are $2^{d-1}-1$ such nodes, so G will be called a total of $2^d - 2$ times while traversing the tree. In addition, there is one invocation of G per output value, which we have not counted in the above analysis, because it is not within the *binary* tree. There are 2^{d-1} output values at depth d of the lookup tree, so this adds 2^{d-1} invocations to our total. Hence, the total number of calls of G required to compute all the output values at depth d (in order) is $2^d + 2^{d-1} - 2$.

Now, we show the ℓ-instance concurrent MIM security of Protocol 4. We require the following technical lemma, which can be seen as a generalization of the leftover hash lemma and has a similar proof.

Lemma 6 ([11]). *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*

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

where $U_{\mathcal{Y}}^1, \ldots, U_{\mathcal{Y}}^\ell$ are ℓ independent and uniformly distributed variables.

We now prove two supporting lemmas, before the main theorem.

Lemma 7. *Protocol 4 is secure against ℓ-instance active attacks.*

Proof. Recall that an ℓ-instance 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 4 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 probablity, all the hash function seeds will be distinct. Therefore, an ℓ-instance 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 $U_{(\ell+1)\cdot n} + \mathbf{a}_1, U_{(\ell+1)\cdot n} + \mathbf{a}_2, \ldots, U_{(\ell+1)\cdot n} + \mathbf{a}_\ell$. Each variable $U_{(\ell+1)\cdot n} + \mathbf{a}_i$ has entropy $H(U_{(\ell+1)\cdot n} + \mathbf{a}_i) = H_\infty(U_{(\ell+1)\cdot n} + \mathbf{a}_i) = (\ell+1)\cdot n$. They are not independent, but they are pairwise different because the a_i's are distinct. Therefore, by Lemma 6,

$$\{h_{\mathbf{s}}, h_{\mathbf{s}}(U_{(\ell+1)\cdot n} + \mathbf{a}_1), \ldots, h_S(U_{(\ell+1)\cdot n} + \mathbf{a}_\ell)\} \stackrel{s}{\approx} \{h_{\mathbf{s}}, U_n^1, \ldots U_n^\ell\}.$$

(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 $\ell = \mathsf{poly}(n)$.)

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.

Lemma 8. *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 6 *Protocol 4 is secure against ℓ-instance concurrent MIM attacks.*

Proof. We show that given an adversary $\mathcal{A}$ which achieves a certain advantage when conducting an ℓ-instance 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 8, $\mathcal{A}$ 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 ℓ-instance concurrent man-in-the-middle attack, there is an equally successful ℓ-instance *active* attack, and finally refer to Lemma 7 for the ℓ-instance 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 ℓ-instance concurrent MIM 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 ℓ-instance *active* attack (since he need not interact with the real verifier $\mathcal{V}$).

Finally, the security of the protocol against ℓ-instance concurrent man-in-the-middle attacks follows from the security against ℓ-instance concurrent active attacks, which was shown in Lemma 7.

E r-UbiAuth is secure against erasures

Theorem 7. r-UbiAuth *is secure against concurrent MIM attacks with erasures.*

Proof (sketch). The security of r-UbiAuth against concurrent MIM attacks *without* erasures is essentially identical to the proof of concurrent MIM security of UbiAuth (Theorem 4). We now sketch why the reinforced lookup function and modified protocol protect against erasures.

First, notice that answers given by an "erased" prover whose secret key has been modified are always rejected. Therefore, the adversary can initially check whether an erasure operation has modified the secret key (with overwhelming probability, this will happen within a constant number of erasure operations), by seeing if the prover's answers continue to be accepted by the verifier after the erasure. However, once an erasure operation has modified the secret key, the verifier will always reject, so the adversary cannot get information about the effectiveness of further erasures from the verifier's decisions. It then remains to argue that the adversary cannot learn to answer correctly by using the *prover's* answers after performing erasure attacks.

By definition of $\mathfrak{R}_n$, after at most n^2 successful erasures (which will happen over $O(n^2)$ attempted erasures with overwhelming probability), the resulting seed will lie outside $\mathfrak{R}_n$ and so the prover will stop answering. Note that this is irreversible: once this happens, the adversary has no way to cause the prover to start answering again. When the adversary attempts an erasure, one of two possible things will happen:

- The erasure will target a bit of the seed which is already 0, so the seed (and induced n-bit seed) will remain unchanged.
- The erasure will change a bit of the seed from 1 to 0. In this case, the induced n-bit seed will change, by definition of r-lookup. More specifically, over the course of n^2 successful erasures, each successful erasure will cause a *completely new n-bit substring* of the n^3-bit seed to be used as the induced n-bit seed. Note that the induced n-bit seeds may have been altered by erasures. However, the important observation is that each "slightly-erased" induced n-bit seed used over the course of n successful erasures is *independent* of every other such slightly-erased induced n-bit seed, since prior to being subjected to erasures, they were independent strings (recall that seeds are sampled from a product distribution $\mathfrak{D}^n$). That is, the prover uses outputs of $\mathsf{lookup}^G_{n,cn}$ from up to n^2 *slightly faulty but independent* n-bit seeds.

The proof is completed by arguing that an adversary who sees PRF output from a series of independent "slightly erased" seeds (but not the corresponding original seeds before erasing) cannot thereby predict PRF outputs on new inputs w.r.t. the original seed.

F Linear-time ℓ-independent hashing

Mansour, Nisan, and Tiwari [29] conjectured that pairwise-independent hash functions cannot be computed in linear time – in particular, that computing them requires $\Omega(n \log(n))$ time. Recently, Ishai et al. [22] disproved this conjecture, and gave a construction of a linear-time computable pairwise-independent hash function. In this section, we show that the [22] construction can be extended to achieve ℓ-wise independence for constant $\ell \in \mathbb{N}$.

Notation. For a vector $v \in \{0,1\}^n$ and a subset of indices $S \subseteq [n]$, we write $v_{[S]}$ to denote the $|S|$-bit vector obtained by restricting v to the coordinates in S. For a tuple of indices $t = (t_1, \ldots, t_d) \in [n]^d$, we write $v_{[t]}$ to denote the d-bit vector $(v_{t_1}, \ldots, v_{t_d})$. We write $||$ for vector concatenation.

Exposure resilient functions [7] (also known as deterministic bit-fixing extractors [9]), are used as a building block in the construction. The definition is given below.

Definition 14. *A function $f : \{0,1\}^n \to \{0,1\}^m$ is an λ-*exposure resilient function *if for any $L \subset [n]$ of size $|L| = n - \lambda$, and for $r \leftarrow \{0,1\}^n$ and $R \leftarrow \{0,1\}^m$ chosen uniformly at random, the distributions $(r_{[L]}, f(r))$ and $(r_{[L]}, R)$ are identical.*

Our construction works as follows. For $n \in \mathbb{N}$, there is an ℓ-wise independent hash function family $\mathcal{H}_n^{C,G,E}$ of functions mapping n bits to n bits. Each family is parametrized by the following:

- $C : \{0,1\}^n \to \Sigma^m$ is the encoding algorithm of an error-correcting code with a constant-size alphabet Σ. The code has minimum distance $c = \Theta(n)$ and constant expansion factor (that is, $m = \Theta(n)$).
- G is a ℓ-wise independent hash function family mapping Σ to Σ.
- $E : \Sigma^m \to \{0,1\}^n$ is an λ-exposure resilient function where $\lambda = \Theta(n)$.

Each function in the family $\mathcal{H}_n^{C,G,E} = \{h_g\}_{g \in G^m}$ is indexed by a vector of hash functions $g = (g_1, \ldots, g_m)$, where each g_i is a member of the (smaller) ℓ-wise independent hash function family G. To sample a hash function in $h_g \leftarrow \mathcal{H}_n$, simply sample M small hash functions $g_1, \ldots, g_m \leftarrow G$.

Each hash function h_g is computed as follows.

1. Encode the input $x \in \{0,1\}^n$ to obtain codeword $y = C(x) \in \Sigma^m$.
2. For each $i \in [m]$, let $z_i = g_i(y_i) \in \Sigma$.
3. Let $z \in \Sigma^m$ denote the concatenation of all z_i, that is, $z = z_1||z_2||\ldots||z_m$. The output of the hash function is $E(z)$.

We remark that this procedure for computing the hash function is the same as in the [22] construction, except that in their work, G is a family of pairwise (rather than ℓ-wise) independent hash functions.

Theorem 8 (ℓ-wise independence of $\mathcal{H}_n^{C,G,E}$). *Let $\ell \geq 2$ be any constant. For any $m = \Theta(n)$, there exist $\lambda, c = \Theta(n)$ such that if C is an error-correcting code with minimum distance c and codeword length m, and E is a λ-exposure resilient function, then $\mathcal{H}_n^{C,G,E}$ is a family of ℓ-wise independent hash functions.*

Proof. Let $c = \Theta(n)$ be the following:

$$c = m - \frac{2m}{\ell(\ell-1)+1}. \tag{1}$$

Note that the right-hand side is $\Theta(n)$, because $m = \Theta(n)$ and $\ell = O(1)$. Moreover, since $\ell > 2$, inequality 1 implies that $0 < c < m$ as required.

Let $x_1, \ldots, x_\ell$ be any distinct vectors in $\{0,1\}^n$, and let $h_g \leftarrow \mathcal{H}_n$ be a hash function sampled from the family $\mathcal{H}_n^{C,G,E}$. Let the set of corresponding codewords be denoted by $Y = \{C(x_1), \ldots, C(x_\ell)\}$.

Define the *overlap* of two codewords $y, y' \in \Sigma^m$ as the set of positions $k \in [m]$ for which the k^{th} elements are equal: that is, where $y_k = y'_k$. Formally,

$$\mathsf{Overlap}(y, y') = \{k \in [m] \ : \ y_k = y'_k\}.$$

Building on this, we define the set $\overline{L}$ to be:

$$[m] \setminus \bigcup_{y,y' \in Y, y \neq y'} \mathsf{Overlap}(y, y') = \{k \in [m] \ : \ \forall y, y' \in Y, \ y_k \neq y'_k\}.$$

In other words, $\overline{L}$ is the set of positions k for which the k^{th} elements of of the codewords in Y are pairwise distinct.

Due to the minimum distance of the error-correcting code,

$$|\mathsf{Overlap}(C(x), C(x'))| \leq m - c$$

for all distinct $x, x' \in \{0,1\}^n$. Then, since $|Y| = \ell$:

$$\left| \bigcup_{y,y' \in Y, y \neq y'} \mathsf{Overlap}(y, y') \right| \leq (m-c) \cdot \frac{\ell \cdot (\ell-1)}{2} \tag{2}$$

$$\leq m. \tag{3}$$

Inequality 3 follows by substituting equation 1 into inequality 2. Moreover, the right-hand side of inequality 2 is clearly $\Theta(m)$ since $c = \Theta(m)$ and ℓ is constant. From this, it follows that

$$\left|\overline{L}\right| = m - \left| \bigcup_{y,y' \in Y, y \neq y'} \mathsf{Overlap}(y, y') \right| = \Theta(m) \ \text{ and } \ \left|\overline{L}\right| > 0. \tag{4}$$

Recall that for each position $k \in \overline{L}$, it holds that the k^{th} elements of of the codewords in Y are pairwise distinct. Then, since g_k is an ℓ-wise independent hash function, the ℓ hash function outputs $g_k(C(x_1)), \ldots, g_k(C(x_\ell))$ will be independent and uniformly distributed. Moreover, since the hash function $g_1, \ldots, g_k$ are chosen independently, the following set consists of independent and uniformly distributed elements:

$$\left\{ g_k(C(x_1)), \ldots, g_k(C(x_\ell)) \ : \ k \in \overline{L} \right\}. \tag{5}$$

Recall that when computing $h_g(x_i)$ (in step 3 of the description above), the input to E is the concatenation of the m "small hashes", $g_1(C(x_i))|| \ldots ||g_m(C(x_i))$. Let z_i denote $g_1(C(x_i))|| \ldots ||g_m(C(x_i))$. Define $L = [m] \setminus \overline{L}$ and $\lambda = |L| = \Theta(m)$. For each input $x_i \in \{0,1\}^n$ to the hash function h_g, E is evaluated on an input z_i for which $(z_i)_{[\overline{L}]}$ is distributed independently at random (this follows from 5). Hence, by the λ-exposure resilience of E, the outputs $E(z_1), \ldots, E(z_\ell)$ are distributed independently and randomly. The theorem follows.

Finally, we show that the hash functions in $\mathcal{H}_n^{C,T,G,E}$ can be computed and sampled in linear time.

Theorem 9. *If C and E are computable in linear time, then each hash function $h_g \in \mathcal{H}_n^{C,G,E}$ can be computed in linear time. Moreover, sampling a hash function $h_g \leftarrow \mathcal{H}_n^{C,G,E}$ can be done in linear time.*

Proof. Since C is linear-time computable, step 1 is computable in linear time, and has a linear-size output $y \in \{0,1\}^m$. In step 2, many small hashes of the form $g_i(y_i)$ are computed, where $g_i \leftarrow G$. Since the input to the hash function g_i is of constant size, each such evaluation of g_i will take constant time. The total number of small hashes computed is $m = \Theta(n)$ which is linear, so step 2 takes linear time. Finally, since E is computable in linear time, and is evaluated on a linear-size input $z \in \{0,1\}^M$, step 3 also takes linear time.

The sampling of a hash function $h_g \leftarrow \mathcal{H}_n^{C,G,E}$ consists of sampling m hash functions $g_1, \ldots, g_m \leftarrow G$. Since the family G is of constant size, each g_i can be sampled in constant time, and there are linearly many of them, so the whole sampling process takes linear time.

There are known constructions of linear-time computable functions that satisfy the properties required by C and E (for any constant ℓ):

- Guruswami and Indyk [19] construct error correcting codes which have linear-time encoding (and decoding) algorithms, and for any positive constant $\varepsilon < 1$, their construction can achieve minimum distance c such that $c/m = \varepsilon$ with a constant-factor expansion. The encoding function of these codes would be suitable for use as C.
- Ishai et al. [22] give a construction of an infinite family of λ-exposure resilient functions mapping n bits to m bits, where $\lambda = \Theta(n)$ and $m = \Theta(n)$.