Forward-Secure Encryption with Fast Forwarding

Yevgeniy Dodis[1]*, Daniel Jost[1]**  , and Harish Karthikeyan[2]***

[1] New York University, New York, USA
{dodis,daniel.jost}@cs.nyu.edu
[2] J.P. Morgan AI Research, New York, USA
harish.karthikeyan@jpmchase.com

Abstract. Forward-secure encryption (FSE) allows communicating parties to refresh their keys across epochs, in a way that compromising the current secret key leaves all prior encrypted communication secure. We investigate a novel dimension in the design of FSE schemes: fast-forwarding (FF). This refers to the ability of a stale communication party, that is "stuck" in an old epoch, to efficiently "catch up" to the newest state, and frequently arises in practice. While this dimension was not explicitly considered in prior work, we observe that one can augment prior FSEs — both in symmetric- and public-key settings — to support fast-forwarding which is sublinear in the number of epochs. However, the resulting schemes have disadvantages: the symmetric-key scheme is a security parameter slower than any conventional stream cipher, while the public-key scheme inherits the inefficiencies of the HIBE-based forward-secure PKE.

To address these inefficiencies, we look at the common real-life situation which we call the *bulletin board model*, where communicating parties rely on some infrastructure — such as an application provider — to help them store and deliver ciphertexts to each other. We then define and construct FF-FSE in the bulletin board model, which addresses the above-mentioned disadvantages. In particular,

- Our FF-stream-cipher in the bulletin-board model has: (a) *constant* state size; (b) *constant* normal (no fast-forward) operation; and (c) *logarithmic* fast-forward property. This essentially matches the efficiency of non-fast-forwardable stream ciphers, at the cost of constant communication complexity with the bulletin board per update.
- Our public-key FF-FSE avoids HIBE-based techniques by instead using so-called updatable public-key encryption (UPKE), introduced in several recent works (and more efficient than public-key FSEs). Our UPKE-based scheme uses a novel type of "update graph" that we construct in this work. Our graph has constant in-degree, logarithmic diameter, and logarithmic "cut property" which is essential for the efficiency of our schemes. Combined with recent UPKE schemes, we get two FF-FSEs in the bulletin board model, under the DDH and the LWE assumptions.

* Partially supported by gifts from VMware Labs and Algorand Foundation, and NSF grants 1815546 and 2055578.
** Research supported by the Swiss National Science Foundation (SNF) via Fellowship no. P2EZP2_195410.
*** Work done while at New York University, New York, USA.

Table of Contents

1 Introduction

Forward Secrecy. Encryption is the fundamental building block of cryptography designed to protect the confidentiality of data such as messages. The security of encryption is, however, confined by its requirement to secretly store the keys. Indeed, leaking an encryption scheme's secret key material typically means that all security is forgone. With cryptographic applications nowadays also typically running on a wide variety of different (and often poorly maintained) devices, alongside other software outside of the control of the cryptographic engineer, such key exposures pose a very real threat scenario.

This risk can be partially mitigated by *forward security*, which refers to the concept that the corruption of a system at some point should not adversely affect the security of prior operations. While initially proposed as a concept for key exchange [34,23] it soon got broadened to incorporate a variety of non-interactive cryptographic primitives, such as forward-secure public-key encryption [18] and forward-secure signatures [7,42]. Roughly speaking, the non-interactive notions share the idea that they divide time into *epochs* with the objective that leaking the secret state at epoch i does not endanger the security properties of past epochs $j < i$.

Fast-Forwarding. In this work, we investigate a novel dimension of the price — in terms of computational and storage overhead — of forward-secure encryption: *fast-forwarding*. The term fast-forwarding refers to the ability of a stale communication party, that is "stuck" in an old epoch, to efficiently "catch up" to the newest state. Such a situation, for instance, might occur if the user has a device that is only sporadically used and has consequently been turned off over a prolonged period.

Indeed, for many applications recovering the latest (or a recent) state seems of intrinsically higher priority than recovering the intermediate states. For example, in an email or a group chat it is often the case that messages sent weeks or months ago might simply no longer be relevant while replying to a recent conversation might be urgent. Moreover, in many communication protocols *sending* messages requires first obtaining (reasonably) up-to-date key material, further motivating the need for fast-forwarding. An example would be secure group messaging (such as MLS) where the group maintains a shared symmetric key that is distributed using public-key cryptography, which is then used to symmetrically encrypt and authenticate messages. It has been proposed to strengthen MLS' forward secrecy [3] by replacing the PKE used for distributing those group keys with a variant of FS-PKE (called UPKE). One of the main drawbacks of that proposal, however, was the lack of fast-forwarding, resulting in a party stuck in an old state having to restore the latest group keys in linear time before being able to send any message. While sequentially downloading old messages might be fine, being unable to send messages — until that process is completed — could trigger an assortment of problems.

1.1 Basic Solutions and A New Dimension

The functionality of forward-secure encryption — either symmetric- or public-key — automatically ensures that one can (fast-)forward from period i to period $j \gg i$ in time proportional to $(j - i)$. In this work, we will call such solutions *linear* and ask if one can build forward-secure encryption with a *sublinear* fast-forward property. To the best of our knowledge, this question was not explicitly considered in the literature. However, one can look at existing techniques for ensuring forward security and come up with some initial observations and solutions.

Symmetric Encryption: Stream Ciphers. Forward-secure symmetric-key encryption is typically achieved using basic stream ciphers, constructed from iteratively evaluating a pseudorandom generator (PRG) [8]. While this construction is efficient, as it only requires a constant size state and a constant number of cryptographic operations to encrypt the next message, it does not allow fast-forwarding: if the receiver last decrypted ciphertext i and now gets ciphertext $j \gg i$, then they need to advance the underlying PRG by $(j - i)$ steps. The existence of an efficient fast-forwarding method would be highly surprising for any widely used stream cipher, as they are not based on number theory.[3]

Instead, we can observe that the Goldreich-Goldwasser-Micali (GGM) construction can be turned into a forward-secure PRG with the fast-forwarding property. More concretely, one can adapt the template for building forward-secure signature schemes [7,42], where the PRG outputs correspond to the GGM tree's leaves and store the current leaf's right sibling path, i.e., the set of nodes from which exactly all leaves to the right of the current leaf can be deduced. We outline this in more detail in Appendix A.1.

[3] The number-theoretic Blum-Blum-Shub PRG [9] can be modified to allow sublinear (in fact, logarithmic) fast-forwarding by additionally keeping the factorization of $N = pq$. Doing so, however, loses forward security.

Lemma 1 (Informal). *The template [7,42] (for building forward-secure signature schemes) can be adapted to the Goldreich-Goldwasser-Micali (GGM) construction [32] to build a forward-secure PRG with the fast-forward property. If n denotes the maximal number of epochs, then the scheme stores $\mathcal{O}(\log(n))$ seeds as local state, and sequential updating as well as fast-forwarding from epoch i to $j > i$ take $\mathcal{O}(\log(n))$ PRG expansions.*

While practically efficient, this folklore construction comes at the cost of worst-case logarithmic sequential evaluations, logarithmic local state, as well as a priori bounded number of overall evaluations. While some of those restrictions can be circumvented using more elaborate constructions — such as growing trees or potentially a cleverly designed amortized evaluation strategy — at least logarithmic-sized storage seems inherent. Thus, wearing a theoretician's hat, the first question we ask is:

Question 1. Can one have a model that allows for fast-forward stream ciphers simultaneously having: (a) *constant* state size; (b) *constant* sequential (no fast-forward) operation; and (c) sublinear (ideally, *logarithmic*) fast-forward property?

Forward-Secure Public-Key Encryption. In the public-key setting, Canetti, Halevi, and Katz [18] introduced the notion of *forward-secure public-key encryption* (FS-PKE) and presented a generic construction of FS-PKE from hierarchical identity-based encryption (HIBE). Their construction essentially mirrors the simple logarithmic construction of the fast-forward PRG mentioned above, but replaces the "GGM tree" with the "HIBE tree." As a result, we observe[4] that this construction allows us to fast-forward from any epoch i to any epoch $j > i$ using $\mathcal{O}(\log j)$ many HIBE secret-key expansions, needs logarithmic-sized storage of HIBE keys (which, in turn, might be long, depending on the HIBE used) and does worst-case logarithmic many secret-key expansions to just proceed to the next epoch.

As of today, this generic construction from HIBE remains the only non-trivial FS-PKE known. Unfortunately, while HIBE schemes from various assumptions exist [10,20,11,28,27,16], they are all either built from primitives not readily available in widespread cryptographic libraries (e.g., bilinear maps) or are primarily theoretical. To the best of our knowledge, this plays a significant role in why FS-PKE has never gained significant adoption in the real world. Hence, we ask:

Question 2. Can one have a meaningful model for fast-forward public-key encryption that potentially enables more efficient schemes than the generic HIBE-based solution mentioned above?

Bulletin Board Model. To address our motivating questions above, we notice that most secure real-world communication applications critically rely on the existence of some centralized server, whose job is to store and appropriately route encrypted messages to the corresponding participants. In other words, since communicating parties might not all be online, or might not have direct communication channels among them, one anyway has to implement some mechanism where old ciphertexts will be delivered to parties when those parties come online and request them. In practice, those servers often perform additional tasks such as helping people discover each other's keys, verifying the authenticity of the keys, serializing the order of concurrently received messages, etc.

End-to-end (E2E) security means that this centralized infrastructure is treated as untrusted, ensuring that a breach or a subpoena cannot affect the users' security. For most applications, however, the server's collaboration is required for correctness.[5] Generally speaking, for such server-assisted protocols, one requires that the most harm a malicious server can inflict is a denial of service (DoS) attack. This is typically deemed an acceptable risk, as a DoS attack is against the service provider's economic incentives.

In our work, we assume the existence of such a server and abstract it as a *bulletin board* functionality. Intuitively (see Section 3.1), this functionality allows all the parties to append some data to a public board, in a way that this data is automatically serialized (so that everybody reads it in the same order), and cannot disappear. Moreover, while the size of the bulletin board is allowed to grow linearly with the number of epochs, it does not "count" toward any of the parties' storage. However, it is also not completely "free", as any party's reading/appending to the bulletin board counts toward the efficiency of this party. More concretely, for primitives using this bulletin-board model, we consider the communication complexity (both in the size of transmitted

[4] For completeness we present the scheme in Appendix A.2

[5] Indeed, many secure messaging systems are designed for the rather peculiar model where the server is assumed to be somewhat-but-not-fully trusted. For instance, MLS aims to provide E2E security, yet exhibits weaknesses if the server does not consistently order messages.

messages as well as required rounds) as part of the respective efficiency notion but choose to disregard the server's storage requirements. This is motivated by real-world communication applications often treating server storage as essentially free (at least for "short" messages such as control messages or text messages, but not necessarily multi-media content) while paying close attention to the efficiency constraints of end-user devices. While disregarding server storage is an oversimplification, we remark that purging could be done in practice, at a potential functionality loss.[6] We can now make our Questions 1 and 2 more precise. Namely, we will ask (and answer) them *in the bulletin-board model*:

Question 3. Assuming the existence of a bulletin board, can we design forward-secure fast-forward stream ciphers and public-key encryption schemes satisfying the efficiency requirements stated in Questions 1 and 2, respectively?

It is prudent to point out that regular forward-secure stream ciphers are already extremely efficient. The bulletin board, however, will help us achieve the fast-forward property whose study we initiate. Even with fast-forwarding, however, the bulletin board model may be primarily of theoretical interest, as the adapted GGM construction is most likely efficient enough for all practical purposes, and the communication latency with the bulletin board likely outweighs the reduced local storage requirement. Nevertheless, we view Question 1 as an interesting open theoretical problem, as even a solution with fast-forwarding and either constant storage or constant sequential updates is an open problem.

In contrast, in the public-key setting there exists no truly practical[7] FS-PKE. It also appears that the bulletin board does not offer any benefit for the HIBE-based FS-PKE schemes. However, for its variant known as *Updatable Public-Key Encryption* (UPKE) [38,3], the bulletin board provides some critical functionality support. We discuss this in more detail below to motivate our new notion of *fast-forward UPKE*.

1.2 Our Contributions

Modeling. We provide a simple yet powerful formalization of the *bulletin board model* capturing a central server offering shared untrusted storage to assist the protocol execution. Our model entirely avoids complications such as interactive models of computation, is carefully designed to ensure that it does not introduce any side-effects such as the access pattern leaking secret information, and allows to easily capture bandwidth constraints.

We then provide rigorous definitions of two forward-secure encryption primitives in the bulletin board model, using a common template: *fast-forwardable stream cipher* and *fast-forwardable updatable public-key encryption* (FF-UPKE). We define correctness as well as IND-CPA security for both notions. We stress that our notions are the first formalization of the fast-forwarding property: while we observe that GGM-PRF and HIBE-based FS-PKE happen to allow for such an operation, none of the respective notions mandates/formalizes it.

Fast-Forwardable Stream Cipher. As a first scheme, we present a fast-forwardable stream cipher, in the bulletin board model, that requires *constant* (in the number of epochs) storage and a *constant* number of cryptographic operations to sequentially advance to the next epoch while allowing to fast-forward to any epoch j in $\mathcal{O}(\log j)$ cryptographic operations. The communication bandwidth of each operation also coincidences with the number of cryptographic operations mentioned above. Thus, we answer the first part of Question 3 affirmatively. Our construction is based on carefully adapting the GGM-based forward-secure stream cipher to:

(1) avoid the logarithmic worst-case computational complexity by appropriate amortization;
(2) offload most of the local storage to the bulletin board without compromising forward secrecy.

Roughly speaking, our scheme expands two GGM nodes per sequential update to ensure that whenever we need a leaf it has already been expanded. As this leads to linearly many expanded but not yet consumed seeds, we have to properly outsource to the bulletin board. This is achieved by encrypting them under independent keys associated with the GGM nodes (also derived from the parent seed), over time forming a linked list among the leaves in increasing order. In the meantime, forward-secrecy is preserved as all encryptions obey the tree's preorder, i.e., we only encrypt a node v's seed under nodes with a smaller preorder index. Let us briefly remark on the potential use-cases of such a solution. Compared to the folklore GGM-based solution, in most settings,

[6] These concerns are interesting but orthogonal to our contributions.
[7] Despite some remarkable progress in the construction of pairing-based HIBE (e.g. [11]) over the last decades, those solutions have never gained any widespread adoption, partially for their omission from popular cryptographic libraries.

trading logarithmic local computation and storage overhead for the need for communication appears to be highly undesirable. However, when used e.g. in the context of secure messaging, the party has to communicate with the server anyway, and as such our construction does not incur a cost in terms of round-trips but merely bandwidth. In such cases, trading a small bandwidth overhead for a logarithmic computation gain could be worthwhile.

One might attempt to apply the same techniques to the HIBE-based FS-PKE construction. We observe, however, that they do not directly translate over, as the senders cannot help the (typically single) receiver. As a result, while the respective solution inherently does support logarithmic fast-forwarding (without even using the bulletin board) supporting constant-time sequential updates does not work in the same way as in the symmetric setting. Concretely, when using a bulletin board to amortize sequential updates (by outsourcing encryptions of secret keys) only receivers knowing those secret keys can upload the respective ciphertexts. (Note that the security of outsourcing an encryption of a HIBE secret key under an earlier public key is not implied by standard security notions, but should hold for most practical schemes.) This has two major drawbacks. First, for a setting with a single receiver, once the receiver fast-forwards to obtain the decryption key of an epoch $j \gg i$, they would be "stuck" there and have to make up the missed $(j - i)$ sequential operations to "complete" the bulletin board before being able to sequentially update in a constant number of operations again. Second, for certain applications where all receivers are assumed to be only sporadically online, having receivers to maintain the bulletin board is generally undesirable. We thus focus on the slightly different primitive of updatable public-key encryption — which is well-suited for the bulletin-board model — instead.

Detour: Updatable Public-Key Encryption. Motivated by various applications to secure messaging, forward-secure PKE in a setting where an untrusted server provides synchronization among parties has been considered under the name UPKE in the literature [38,3]. The idea of UPKE is that one can use ciphertexts — conveniently serialized and placed on the bulletin board (abstracting the messaging server) — to also contain information on how to move from the old $(\mathsf{pk}, \mathsf{sk})$ tuple, into a new $(\mathsf{pk}', \mathsf{sk}')$, in a way that: (a) new messages will be encrypted by pk' and decrypted by sk'; (b) exposure of sk' does not help to decrypt prior ciphertexts (including the one just sent under pk); (c) the person preparing ciphertext helps to "move" from sk to sk' without knowing either secret key but pk only. To show the potential of the bulletin board, the work of [38] provided a very simple and efficient UPKE (in the random oracle model) that has similar efficiency to the underlying ElGamal encryption. Recently, [24] also built two standard model UPKE schemes from the DDH and LWE assumptions, which were again much more efficient than their HIBE-based counterparts based on either DDH or LWE.

These constructions further validated the intuition that UPKE may be a significantly cheaper alternative compared to regular FS-PKE. They, however, eschew the inherent fast-forwarding property the generic HIBE-based FS-PKE enjoys. It thus remained an open problem whether it is possible to build a truly efficient fast-forwardable PKE primitive. In this work, we provide a partially affirmative answer for FF-UPKE. While not truly practical, our constructions are again more efficient when compared to their HIBE counterparts from the same assumptions. Specifically, the LWE-based construction is significantly simpler and more efficient than the best-known post-quantum secure FS-PKE scheme (see Section 1.3). Moreover, our novel approach initiates the study in this new dimension and hopefully serves as a launchpad for improved construction. Critically, we propose a generic FF-UPKE construction (much like the original FS-PKE scheme) that is not tethered to any particular assumption. In this process, we introduce a new primitive, whose instantiation directly implies FF-UPKE.

Our Work: FF-UPKE. We define *FF-UPKE*. As with standard UPKE, the sender can create special ciphertexts which do not encrypt messages,[8] but can help move from the current tuple $(\mathsf{pk}_i, \mathsf{sk}_i)$ for epoch i to the next tuple $(\mathsf{pk}_{i+1}, \mathsf{sk}_{i+1})$ for epoch $(i + 1)$. To support fast-forwarding from epoch i to epoch j, we introduce a special "leaping" algorithm. This algorithm can help a receiver[9] who currently holds sk_i to get to the latest key sk_j, by only reading a sublinear (in $(j - i)$) number of messages from the bulletin board, and performing a sublinear number of cryptographic operations. See Definition 5.

Aside from solving the practical problem of allowing an offline receiver to quickly catch up with the current messages, FF-UPKE also weakens the strictly sequential requirement of standard UPKE, that receiver should

[8] For the highest security, these can be sent after every regular ciphertext encrypting a message, but we do not require this to allow for the most flexibility.

[9] Of course, a new sender who "fell behind" other senders, can trivially "catch up" by retrieving the latest public key from the bulletin board.

read and process every previous key update message. Indeed, the sublinear efficiency requirement of FF-UPKE means that the receiver has multiple "opportunities" to catch up, even if it cannot access some of the key update messages.[10]

FF-UPKE Construction. We also present a novel FF-UPKE scheme. To this end, we observe that all existing UPKE schemes refresh the secret key by having the sender choose an "update secret" δ that is then sent encrypted under the current public key pk_i. After decrypting δ using sk_i, the next secret key $\mathsf{sk}_{i+1} = \mathsf{sk}_i + \delta$ (using appropriate group operation $+$), while the sender can compute the next public key pk_{i+1} using only the current public key pk_i and the value δ. Our generic FF-UPKE construction is built around the idea of so-called *cumulative updates* using any so-called *update-homomorphic* UPKE scheme — a notion we introduce to formalize that multiple such update messages can be homomorphically combined. (See Definition 8 for the precise formalization of this requirement.)

As in prior UPKE schemes, in our construction, the sender for epoch i will choose update secret δ_{i+1}, and we will have the invariant that $\mathsf{sk}_j = \mathsf{sk}_i + \Delta[i,j]$, where $\Delta[i,j] \coloneqq (\delta_{i+1} + \ldots + \delta_j)$, for any $j > i$. Now, however, senders can use the update homomorphism to create encryptions $\mathsf{up}_{i,j}$ that "cumulatively encrypt" $\Delta[i,j]$ under pk_i, for certain carefully chosen pairs $i < j$. Those encryption are then stored in the bulletin board to allow the receiver to fast-forward.

Finally, we show that with minor modifications the standard model DDH/LWE UPKE schemes of Dodis et al. [24] both satisfy the above homomorphism.[11]

Technical tool: Update Graph. As one can see, the efficiency of our cumulative update scheme for building FF-UPKE from update-homomorphic UPKE critically depends on the properties of what we call an *update graph* $\mathcal{G}$, which will govern the collection of edges (i,j) for which parties need to maintain update ciphertexts $\mathsf{up}_{i,j}$. As it turns out (see Definition 11), three such parameters will be essential for understanding the efficiency of our construction:

- The *maximum in-degree* $\alpha(n)$ of any vertex $j \leq n$;
- The *diameter* $\beta(n)$ of the sub-graph of the first n vertices;
- The cardinality $\gamma(n)$ of the *active set* that includes all vertices $i < n$ which have at least one edge (i,j) with $j > n$ in $\mathcal{G}$.

We call such graphs $(\alpha(n), \beta(n), \gamma(n))$-*update graphs*. To the best of our knowledge, while many related notions of dynamic graphs are known in the literature (e.g., see [44] and references therein), including graphs having small in-degree and diameter, the exact notion of update graph we need for our construction is new.

We build a nearly optimal $(2, \mathcal{O}(\log n), \mathcal{O}(\log n))$-update graph. Our construction is inspired by the simple family of spanner graphs that recursively join two consecutive graphs of an equal number of nodes with an overarching edge spanning from the first to the last node. We observe that this results in a graph with logarithmic diameter but also logarithmic indegree. To circumvent the growing indegree, we modify this construction slightly and, in each recursive step, add one additional node at the end, to which the overarching edge connects. We call this graph $\tilde{\mathcal{G}}_n$, where n denotes its number of nodes. See Fig. 1 for the example of $\tilde{\mathcal{G}}_{16}$.

As we will see, using this graph $\tilde{\mathcal{G}}$ results in our final FF-UPKE scheme having logarithmic overhead for key update and fast-forwarding, and no overhead for public-key size, encryption, and decryption, as compared to the underlying update-homomorphic UPKE.

Putting It All Together. Instantiating our generic cumulative update scheme with our update graph and the two homomorphic-update UPKE scheme from DDH and LWE, we get two concrete FF-UPKE schemes from DDH and LWE, respectively, achieving greater efficiency than the best-known HIBE-based FS-PKE scheme from the same assumption (see Section 1.3) and answering Question 2 in the affirmative, in the bulletin board model.

[10] Of course, the sender for epoch i should still be able to get the current key pk_i.

[11] Interestingly, the most efficient random-oracle based scheme [38,3] does not appear to be update-homomorphic, and will not be enough for our purposes.

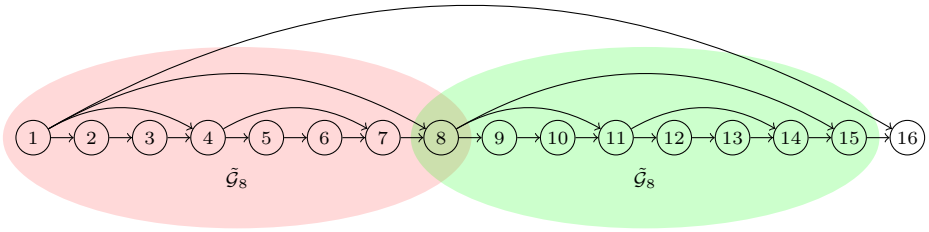

Fig. 1: The update graph $\tilde{\mathcal{G}}_{16}$, consists of two subgraphs $\tilde{\mathcal{G}}_8$ and an extra node.

1.3 Related Work

Hierarchical Identity Based Encryption. As mentioned before, the work of Canetti, Halevi, and Katz [19] showed how to generically construct Forward-Secure Public Key Encryption from Hierarchical Identity Based Encryption [31,35]. Indeed, over time, various HIBE schemes have been proposed under an assortment of assumptions such as LWE [20,2], CDH/DDH [28,27,16], and pairing-based Diffie-Hellman Assumptions [10,11], among others. However, despite the beautiful theory, HIBE-based schemes have not found much adoption in practice, either due to the reluctance of practitioners to use pairings, or because the constructions are rather impractical. For example, the CDH/DDH-based constructions [28,27,16], while giving us constructions of FS-PKE in the standard model, rely on garbled circuits. More formally, if κ is the security parameter, it relies on a chain of $\mathcal{O}(\kappa)$ such circuits, and the blow-up of the public key operations performed by the circuits is significant. HIBE constructions based on LWE [20,2] rely on either lattice trapdoors or GPV-style pre-image sampling [30] which are inefficient and quite complex. Additionally, all of these HIBE constructions suffer from overhead of $\mathcal{O}(\kappa)$ to support an unbounded number of time periods in the FS-PKE construction built from HIBE. As the result, while our new DDH/LWE schemes are not yet as practical as PKEs from the same assumption (and we also did not try to optimize all the constants in our constructions for the elegance of presentation), they certainly are more efficient than the corresponding HIBE-based FS-PKEs, even taking into account the reliance on the bulletin board model.

Forward-Secure Signatures. Anderson [5] first proposed the idea of Forward-Secure signatures. The idea was that the compromise of a secret key at a time i should not allow for the forgery of messages at a time $j < i$. Construction of this primitive was proposed by Bellare and Miner [7] and later extended and improved by Malkin *et al.* [42]. There are still other constructions that are secure in the random oracle model [1,36,40].

Key Evolving Encryption Schemes. Two recent works - Jaeger and Stepanovs [37] and Poettering and Rössler [43] - proposed a scheme that was secure even when the key updates were labeled by arbitrary (even adversarially chosen) strings. This is a stronger setting than even FS-PKE and unsurprisingly, these constructions were realized from HIBE Schemes.

Updatable Public Key Encryption. The work of Jost *et al.* [38] and Alwen *et al.* [3] formally introduced the primitive and presented constructions that were secure in the random oracle model. The work of Dodis *et al.* [24] explored constructions that were secure in the standard model. In addition, they also considered extensions of the simpler CPA-based security definition to stronger definitions.

Updatable Encryption. Updatable Encryption [13,29,41,39,15,12], vastly different from the idea of Updatable Public Key Encryption, explores the orthogonal problem of updating the ciphertext that was encrypted under a key at a time i to be consistent, i.e., decryptable by the key at a time $j > i$. This primitive, however, is for the *symmetric key* setting. Informally, the construction produces different ciphertexts for the same messages under different keys. The goal is to produce tokens that help update the ciphertext without revealing any information about the underlying message.

Key Insulated Public Key Cryptosystems. Key Insulated Public Key Cryptosystems [25,26], though motivated for other purposes, achieves the feature of fast-forwardability. They do this by offering random-access key updates which help move from the current period i to any other period j. However, the constructions and indeed the setting assume that there is a party that is trusted, resistant to exposure/leakage, and has a secure channel to the secret key owner.

Puncturable (Public Key) Encryption. Puncturable Encryption [6,45] and Puncturable PKE [33,22,46] achieve forward secrecy on a per ciphertext basis. That is, puncturing has to work purely based on received ciphertexts, rather than dividing time into epochs, with senders not having to "target" a certain epoch or obtain an updated (public) key. Hence, puncturable PKE solves a much more difficult problem than FS-PKE or fast-forwardable UPKE, leading to *significantly less efficient* solutions.

Puncturable PRFs. Our fast-forwardable PRG construction is quite similar to some of the constructions of puncturable PRFs based on the GGM construction [14]. We remark, however, that the (standard) *notion* of a puncturable PRF is quite different and, for example, lacks the iterative aspect of "continuously re-puncturing" which we require for our notion.

2 Preliminaries

Notation. We write $\mathbb{N} \coloneqq \{1, 2, \ldots\}$ and for $x \in \mathbb{N}$ we write $[x] \coloneqq \{1, 2, \ldots, x\}$. We write $x \leftarrow a$ to assign the value a to the variable x. Moreover, for a set $\mathcal{S}$ we write $x \leftarrow\!\!\$\ \mathcal{S}$ to denote sampling an element from $\mathcal{S}$ uniformly at random or according. For a probabilistic algorithm A, we write $\mathsf{A}(\cdot; r)$ to denote that A is run with explicit randomness r. The security parameter is denoted by κ.

Graphs. A directed graph $\mathcal{G} = (V, E)$ consists of a vertices set V and an edge set $E \subseteq V^2$. For an edge $(u, v) \in E$, we call u the *tail* and v the *head*. For a node $w \in V$, we denote by $E^{\mathsf{in}}_{\mathcal{G}}(w) \coloneqq \{(u, v) \in E \mid v = w\}$ the set of incoming edges, and by $E^{\mathsf{out}}_{\mathcal{G}}(w) \coloneqq \{(u, v) \in E \mid u = w\}$ the set of outgoing edges, respectively. Moreover, we denote by $\deg^{\mathsf{in}}_{\mathcal{G}}(w) \coloneqq |E^{\mathsf{in}}_{\mathcal{G}}(w)|$ and $\deg^{\mathsf{out}}_{\mathcal{G}}(w) \coloneqq |E^{\mathsf{out}}_{\mathcal{G}}(w)|$ the in- and outdegrees, respectively. We often omit to specify the graph for these functions when the context is clear. For $u, v \in V$ we say that edges $\big((v_1, v_2), (v_2, v_3), \ldots, (v_n, v_{n+1})\big) \in E^n$ is a *path* of length n from v_1 to v_{n+1}. If the concrete path is not of relevance, we sometimes use $u \rightsquigarrow v$ as a shorthand notation for and refer by $|u \rightsquigarrow v|$ to its length. Additionally, we refer to $d(u, v)$ as the minimum length of all paths from u to v (or ∞ if no such path exists). Finally, the diameter of the graph $\mathcal{G}$ refers to the maximal distance between any nodes u and v for which a path exists, i.e., $\mathrm{diam}(\mathcal{G}) \coloneqq \max\{d(u, v) \mid u, v \in V \wedge d(u, v) \neq \infty\}$.

Cryptographic Primitives. We make use of a *pseudo-random generator* with expansion factor 4, which is a function $\mathsf{PRG.Expand}\colon \{0,1\}^\kappa \to \{0,1\}^{4\kappa}$ such that the two distribution ensembles $\big\{\mathsf{PRG.Expand}(\mathsf{s}) \mid \mathsf{s} \leftarrow\!\!\$\ \{0,1\}^\kappa\big\}_{\kappa\in\mathbb{N}}$ and $\big\{\mathsf{k} \leftarrow\!\!\$\ \{0,1\}^{4\kappa}\big\}_{\kappa\in\mathbb{N}}$ are computationally indistinguishable.

Moreover, we use a *nonce-based symmetric encryption scheme*, which is a tuple of deterministic algorithms $(\mathsf{SE.Enc}, \mathsf{SE.Dec})$, such that for any encryption $\mathsf{c} \leftarrow \mathsf{SE.Enc}(\mathsf{k}, \mathsf{m}, \mathsf{n})$, the corresponding decryption recovers the correct message $\mathsf{m} \leftarrow \mathsf{SE.Dec}(\mathsf{k}, \mathsf{c}, \mathsf{n})$. We require the scheme to satisfy standard IND-CPA security as long as the nonce n is not reused.

3 Fast-Forwarding in the Bulletin Board Model

3.1 Bulletin Board

In this work, we consider a setting where parties can make use of an append-only bulletin board BB to store and retrieve (shared) information, reducing their storage and computation costs. Intuitively, BB can be thought of as an associative array where for an index $\mathsf{idx} \in \mathcal{I}$ they can retrieve a value $v \leftarrow \mathsf{BB}[\mathsf{idx}]$ either returning the previously stored value v or a special error symbol $\perp$. Along the same lines, $\mathsf{BB}[\mathsf{idx}] \leftarrow v'$ sets the value to v' if it has been previously undefined, or ignores the new value.

We formalize this by using a partial function $\mathsf{BB}\colon \mathcal{I} \rightharpoonup \mathcal{V}$. Moreover, we define two additional operators on the bulletin board: restriction and appending. For a subset of possible indices I, $\mathsf{BB}{\restriction_{\mathrm{I}}}$ denotes the modified bulletin board only defined for those indices. We will use this operation as a convenient way to handle a party "fetching" a subset of the values of the bulletin board alongside the associated indices. The append operations append another bulletin board to the existing one while ignoring all already defined values. We will use this operation to represent a party "uploading" new values to the bulletin board.

Definition 1. *For an index space $\mathcal{I}$ and a value space $\mathcal{V}$, we call a partial function $\mathsf{BB}\colon \mathcal{I} \rightharpoonup \mathcal{V}$ a bulletin board. That is, $\mathsf{BB} \subset \mathcal{I} \times \mathcal{V}$ such that for all $\mathsf{idx} \in \mathcal{I}$ and $v_1, v_2 \in \mathcal{V}$, $(\mathsf{idx}, v_1) \in \mathsf{BB}$ and $(\mathsf{idx}, v_2) \in \mathsf{BB}$ implies $v_1 = v_2$. For a set of indices $\mathrm{I} \subseteq \mathcal{I}$, we denote by $\mathsf{BB}{\restriction_{\mathrm{I}}}$ function restriction. That is, $\mathsf{BB}{\restriction_{\mathrm{I}}} \coloneqq \{(\mathsf{idx}, v) \in \mathsf{BB} \mid \mathsf{idx} \in \mathrm{I}\}$. Moreover, for bulletin boards BB_1 and BB_2, we define $\mathsf{BB}_1 +\!\!+ \mathsf{BB}_2 \coloneqq \mathsf{BB}_1 \cup \mathsf{BB}_2{\restriction_{\mathcal{I}\setminus\mathrm{dom}(\mathsf{BB}_1)}}$, where* dom *denotes the domain.*

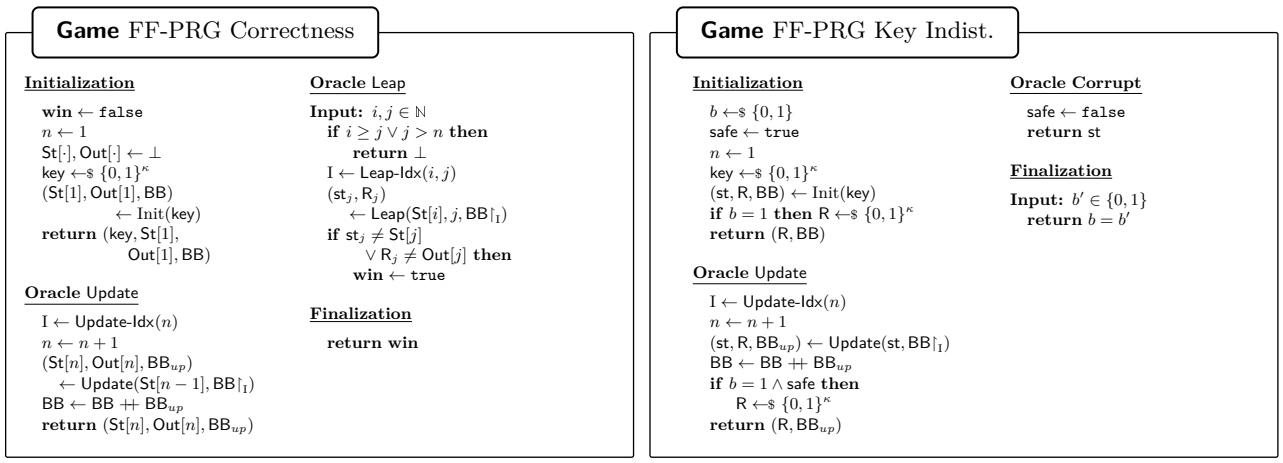

Fig. 2: The correctness (left) and security (right) games of FF-PRGs.

3.2 Fast-Forwardable Stream Ciphers

We investigate the construction of fast-forwardable forward-secure stream ciphers in the bulletin-board model. It is easy to see that the folklore stream cipher construction from a PRG yields this as long as the PRG is both (1) forward secure and (2) fast forwardable. We, thus, focus on fast-forwardable PRGs[12] instead.

Definition 2. *A* Fast-Forwardable PRG (FF-PRG) *consists of the deterministic algorithms* $(\mathrm{Init}, \mathsf{Update}, \mathsf{Update\text{-}Idx}, \mathsf{Leap}, \mathsf{Leap\text{-}Idx})$*, where:*

- *The* $(\mathsf{st}_1, \mathsf{R}_1, \mathsf{BB}_{init}) \leftarrow \mathrm{Init}(\mathsf{key})$ *algorithm takes* $\mathsf{key} \in \{0,1\}^\kappa$ *and produces an initial state* st_1*, output* R_1*, and initial bulletin board state* BB_{init}*.*
- *The* $(\mathsf{st}_{i+1}, \mathsf{R}_{i+1}, \mathsf{BB}_{up}) \leftarrow \mathsf{Update}(\mathsf{st}_i, \mathsf{BB}\restriction_{\mathrm{I}_i})$ *algorithm takes a state and parts of the bulletin board as inputs, and produces the updated state and the next output, as well as content to upload to the bulletin board. The corresponding update-index algorithm* $\mathrm{I}_i \leftarrow \mathsf{Update\text{-}Idx}(i)$ *determines the part of the bulletin board required for this operation.*
- *The* $(\mathsf{st}_j, \mathsf{R}_j) \leftarrow \mathsf{Leap}(\mathsf{st}_i, j, \mathsf{BB}\restriction_{\mathrm{I}_{i,j}})$ *algorithm takes a state* st_i*, the target epoch* $j > i$*, and parts of the bulletin board as inputs, to leap to the* j*-th state and output. The indices are determined by* $\mathrm{I}_{i,j} \leftarrow \mathsf{Leap\text{-}Idx}(i, j)$*.*

Efficiency. For an FF-PRG scheme to be non-trivial we require fast-forwarding to be of sub-linear complexity in $j - i$, concerning both running time and communication complexity. More specifically, we require the output size of $\mathsf{Leap\text{-}Idx}$ to be bounded by fixed polynomials of the security parameter κ, i.e., to be independent of $j - i$.[13] This in turn also implies that the running time of Leap is bounded by a fixed polynomial in κ.

Correctness and Security. For correctness, we intuitively expect that fast-forwarding results in the same output and state as sequentially advancing throughout the epochs. Note, however, that fast-forwarding is meant to be a "catching up" mechanism. Thus, we require fast-forwarding only to work whenever some other party (using the same bulletin board) has already reached the target epoch. This is formalized in Fig. 2 using two oracles: the Update oracle allows to sequentially advance throughout the epochs, while the Leap oracle allows making arbitrary jumps as long as the target epoch has been reached before.

Definition 3 (Correctness). *An FF-PRG is correct if every PPT adversary* $\mathcal{A}$ *has a negligible advantage in winning the correctness game depicted in Fig. 2.*

The key indistinguishability game of an FF-PRG formalizes that R_i look indistinguishable from fresh uniform random outputs. Forward security moreover asserts that for past outputs this holds even once the state is leaked. Note that for defining security, fast-forwarding is irrelevant, as Leap does not write to the bulletin board and, by correctness, results in the same state as sequential updates.

[12] For simplicity, we henceforth omit explicitly mentioning forward secrecy as part of each primitive's name.
[13] Since $j \leq 2^\kappa$, each function in $\mathsf{poly}(\log(j))$ is bounded by a fixed polynomial in κ.

Definition 4 (Security). *A FF-PRG is secure, if every PPT adversary $\mathcal{A}$ has negligible advantage (i.e, $2\Pr[\text{Key-Indist}_{\mathcal{A}} = \texttt{true}] - 1$) in winning the key indistinguishability game depicted in Fig. 2.*

We remark that having explicit indexing algorithms Update-Idx and Leap-Idx that depend on public information only, guarantees that the *access pattern* to the bulletin board does not leak confidential information about a party's state.

Constructions. In Section 4, we present an FF-PRG scheme with the following properties: First, Init and Update perform a constant number of cryptographic operations and output a constant number of elements to be uploaded to the bulletin board, i.e., $|\mathsf{BB}_{init}|, |\mathsf{BB}_{up}| \in \mathcal{O}(1)$. Second, Update only requires a constant number of elements from the bulletin board, i.e., $|\mathsf{Update\text{-}Idx}(i)| \in \mathcal{O}(1)$. Third, all elements on the bulletin board are of size $\mathcal{O}(\kappa)$ (such as a key or an encryption). Finally, $\mathsf{Leap}(\mathsf{st}_i, j, \mathsf{BB}_{in})$ performs at most $\mathcal{O}(\log j)$ operations and $\mathsf{Leap\text{-}Idx}(i, j)$ is of cardinality at most $\mathcal{O}(\log j)$.

Recall from Section 1.2 that while introducing additional communication might often be undesirable (and indeed reducing communication complexity is often a primary goal) there are settings where communication with a centralized server anyways occurs, such as the symmetric ratcheting layer of Signal. There, switching this to our protocol would not add communication latency, but only slightly increased bandwidth. Moreover, our scheme is *concretely efficient* and for 2^T epochs reduces the secret storage compared to the GGM tree by roughly a factor of $T/7$. For example, for $T = 20$ under standard parameter choices of 128 bit seeds we go from 320 bytes to 122 bytes (and the bulletin board material after 2^{20} epochs will be under 50 MB).

3.3 Fast-Forwardable Updatable Public-Key Encryption

We now proceed to formalize fast-forwardable UPKE in the bulletin board model.

Definition 5. *A* Fast-Forwardable Updatable Public-Key Encryption (FF-UPKE) scheme *is a tuple of PPT algorithms* (KeyGen, Encrypt, Decrypt, UpdatePK, UpdatePK-Idx, UpdateSK, UpdateSK-Idx, LeapSK, LeapSK-Idx)*, defined as follows:*

- *The* $(\mathsf{pk}_1, \mathsf{sk}_1, \mathsf{BB}_{init}) \leftarrow \mathsf{KeyGen}(1^\kappa)$ *algorithm outputs an initial secret/public key pair* sk_1 *and* pk_1 *and an initial state of the bulletin board.*
- *The* $\mathsf{c} \leftarrow \mathsf{Encrypt}(\mathsf{pk}_i, \mathsf{m})$ *algorithm encrypts* m *under the public key* pk_i *and the deterministic* $\mathsf{m} \leftarrow \mathsf{Decrypt}(\mathsf{sk}_i, \mathsf{c})$ *algorithm decrypts* c *using the corresponding secret key.*
- *The* $(\mathsf{pk}_{i+1}, \mathsf{BB}_{up}) \leftarrow \mathsf{UpdatePK}(\mathsf{pk}_i, \mathsf{BB}\restriction_{\mathrm{I}_i})$ *algorithm takes a public key and parts of the bulletin board as input, and outputs the updated public key and content to be upload to the bulletin board.*
- *The deterministic* $\mathsf{sk}_{i+1} \leftarrow \mathsf{UpdateSK}(\mathsf{sk}_i, \mathsf{BB}\restriction_{\mathrm{I}_i})$ *algorithm takes a secret key and parts of the bulletin board as inputs, and outputs the updated secret key.*
- *The deterministic* $\mathsf{sk}_j \leftarrow \mathsf{LeapSK}(\mathsf{sk}_i, j, \mathsf{BB}\restriction_{\mathrm{I}_{i,j}})$ *algorithm takes a secret key* sk_i*, the target epoch* $j > i$*, and parts of the bulletin board as inputs.*

The deterministic algorithms $\mathrm{I}_i \leftarrow \mathsf{UpdatePK\text{-}Idx}(i)$, $\mathrm{I}_i \leftarrow \mathsf{UpdateSK\text{-}Idx}(i)$*, and* $\mathrm{I}_{i,j} \leftarrow \mathsf{LeapSK\text{-}Idx}(i, j)$ *determine the part of the bulletin board required for the respective operations.*

Modeling and Efficiency. One of the key properties of UPKE is that UpdatePK may be probabilistic. It is thus assumed that multiple senders coordinate on the advancing of epochs, with only one party executing UpdatePK and then distributing the updated public key to the other senders. (Indeed, this synchronization requirement seems to be what gives UPKE a significant performance lead over FS-PKE.) While in practice this might be achieved by storing the public key on the bulletin board, passing around an explicit public key allows us to easily model that during an epoch parties do not need to access the bulletin board.

Moreover, UpdateSK does not write any information to the bulletin board for the following reasons: First, there is typically only one receiver (per key pair) in a public-key setting, so there is no need to upload information that might help other receivers. Second, if the receiver had to somehow assist senders, this would introduce additional online requirements contradicting the asynchronous nature of public-key communication. (The synchronization among senders to prevent conflicting updates does not require all or any particular of them to be online.)

We require all algorithms except LeapSK to run in polynomial time independent of the epoch i. The LeapSK is allowed to run in time sublinear in $j - i$ (non-triviality). However, we stress that LeapSK-Idx must have a running time, and thus output size, of a fixed polynomial independent of $j - i$, meaning that LeapSK has communication complexity at most $\mathsf{poly}(\log j)$.

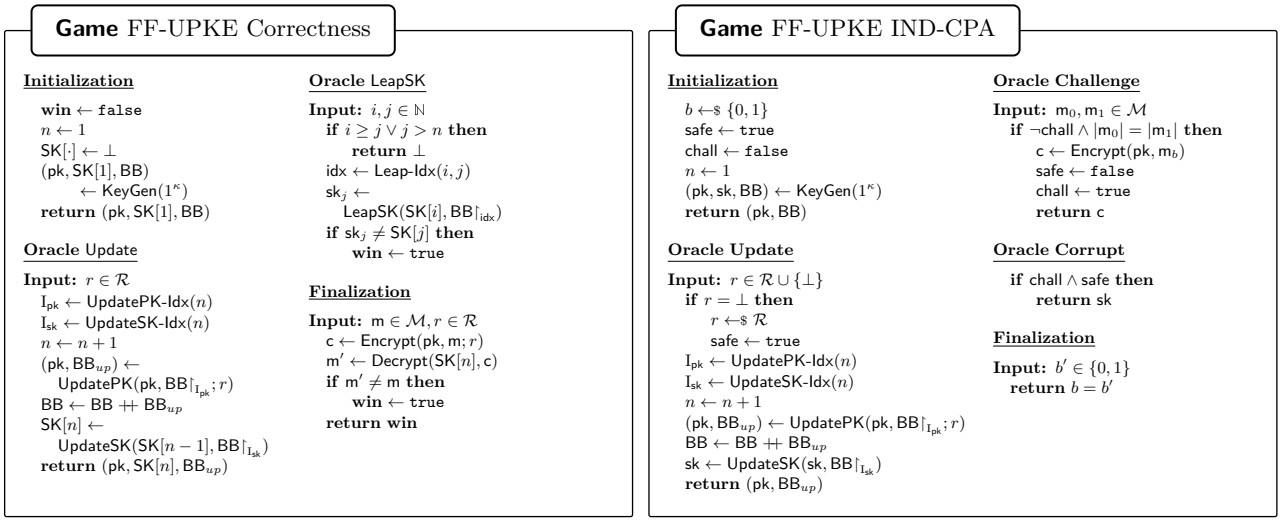

Fig. 3: The correctness (left) and IND-CPA (right) games of FF-UPKE.

Correctness and Security. Correctness is formalized using a game depicted in Fig. 3. In a nutshell, there are two ways for an adversary to break correctness: (1) he breaks the correctness of the encryption, i.e., comes up with a message such that its encryption does not decrypt properly, or (2) he breaks the correctness of the fast-forwarding mechanism. For simplicity, we require from an FF-UPKE scheme that fast-forwarding from epoch i to j results in the same secret key sk_j as would have resulted from sequentially updating. Note that since FF-UPKE is designed for a setting where parties might use bad randomness, the correctness game allows the adversary to choose all randomness.

Definition 6 (Correctness). *An FF-UPKE is said to be correct, if every PPT adversary $\mathcal{A}$ has a negligible advantage in winning the correctness game depicted in Fig. 3.*

Security is formalized as an IND-CPA game, depicted in Fig. 3. The game allows the adversary to make a single challenge from which he must decide whether he received encryption of m_0 or m_1. Ahead, he can make an arbitrary number of updates to the public key, potentially supplying the randomness. Moreover, to formalize forward secrecy, he can corrupt the receiver's state to obtain the secret key — once at least one is secure, i.e., not with adversarially chosen randomness, an update has been applied. Similar to the FF-PRG notion, we observe that the LeapSK algorithm is irrelevant for security.

Definition 7 (Security). *An FF-UPKE is said to be IND-CPA secure, if every PPT adversary $\mathcal{A}$ has a negligible advantage in winning the IND-CPA game depicted in Fig. 3.*

Constructions. In Section 5, we present a generic FF-UPKE scheme where public and secret keys do not grow with the epoch number i, the UpdatePK algorithm reads and writes $\mathcal{O}(\log i)$ positions on the bulletin board, UpdateSK reads $\mathcal{O}(1)$ positions, and LeapSK accesses $\mathcal{O}(\log j)$ position to fast-forward from epoch i to j. The construction makes use of a so-called Update-Homomorphic UPKE scheme as a building block. In Appendix B we provide two concrete instantiations of this building block, both leading two bulletin board values of the order of $\mathcal{O}(\kappa^2)$ many cryptographic elements. In Section 5.4 we moreover discuss some of the barriers encountered in constructing truly efficient FF-UPKE schemes.

4 Constructing a Fast-Forwardable PRNG

In this section, we present a construction of a fast-forwardable PRNG. We first introduce the basic variant supporting a bounded number of epochs. We then extend this construction in Section 4.2 to an unbounded number of epochs.

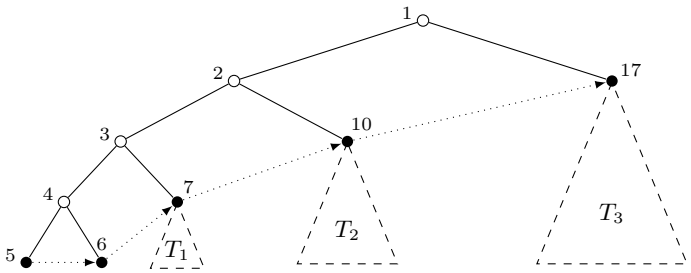

Fig. 4: The tree GGM_4 after the Init algorithm. Dotted arrows represent encryptions outsourced to the bulletin board.

4.1 The Basic Construction

Our construction is based on the GGM construction. To this end, we first observe that in a GGM tree of height h, and thus 2^{h} leaves, there are a total of $2^{\mathsf{h}} - 1$ inner nodes to expand. Hence, the amortized number of expansions over the course of the $2^{\mathsf{h}} - 1$ many possible updates in this tree is just one. In the following, we will show that if for each update we do *two* expansions, then at the time we need a new leaf it has already been derived.

Implemented naively, this would of course make a party's state grow linearly in the number of updates, which is where the outsourcing to the bulletin board comes into play. Roughly speaking, rather than keeping all the expanded seeds in the local state, we encrypt them under an appropriate key to outsource. Those encryption keys are derived from the GGM tree as well. To this end, we modify the expansion step of a node v's seed as follows:

$$(\mathsf{s}_{v_{\text{left}}}, \mathsf{k}_{v_{\text{left}}}, \mathsf{s}_{v_{\text{right}}}, \mathsf{k}_{v_{\text{right}}}) \leftarrow \mathsf{PRG.Expand}(\mathsf{s}_v),$$

where k_v is an encryption key associated with v. Using this key, we can then encrypt another node's seed s_u and key k_u using nonce-based symmetric encryption, i.e., $\mathsf{c} \leftarrow \mathsf{SE.Enc}(\mathsf{k}_v, (\mathsf{s}_u, \mathsf{k}_u), \mathsf{n})$. Concretely, we use the index u as nonce, $\mathsf{n} = u$, and store this ciphertext at index (v, u) in the bulletin board.

To ensure forward secrecy, only certain such links can be stored. Recall to this end that when using the GGM construction as a forward-secure PRNG, one expands the tree's nodes according to the *preorder traversal* and keeps the nodes from the copath (sometimes called sibling path) that are right children as a state. Hence, to preserve forward secrecy, we maintain the following invariant:

I1. Whenever the bulletin board stores an encryption $\mathsf{c} \leftarrow \mathsf{SE.Enc}(\mathsf{k}_v, (\mathsf{s}_u, \mathsf{k}_u), u)$ for nodes u and v, then $\mathsf{preorderIdx}(v) < \mathsf{preorderIdx}(u)$,

where $\mathsf{preorderIdx}(v)$ returns v's index according to the preorder traversal.

Initialization. Let us now turn our attention towards which such links we want to outsource. Initially Init(key) first derives a seed s and outsourcing key k for the root (e.g., $(\mathsf{s}, \mathsf{k}, \cdot, \cdot) \leftarrow \mathsf{PRG.Expand}(\mathsf{key})$ and then proceeds to expand the leftmost path in the GGM. The copath is outsourced to the bulletin board by encrypting each node under the previous when traversing the copath from the leaf to the root. Additionally, we encrypt the first copath node under its left sibling, i.e., the first epoch's leaf. All of those encryptions satisfy Invariant 1.

See Fig. 4 for the example of GGM_4, the GGM tree of height 4, at the end of the Init operation. For clarity, we labeled each node with its preorder index. White nodes represent inner nodes that have already been expanded, black nodes those for which the seeds are currently known, and gray nodes are currently beyond the expansion horizon. The dotted arrows represent the outsourced encryptions, i.e., a dotted arrow from node v to u means that we store $\mathsf{SE.Enc}(\mathsf{k}_v, (\mathsf{s}_u, \mathsf{k}_u), u)$ at position (v, u) in the bulletin board.

The Init algorithm outputs the seed $R_1 = \mathsf{s}_{\mathsf{h}+1}$ (of the leftmost leaf) and a state containing the following values: the key k_i and the seeds and keys of the thirst three nodes on i's right copath, starting at its right sibling. We call those three nodes the initial frontier, which we discuss in a moment. In our example of GGM_4, this is s_5 and $(\mathsf{s}_j, \mathsf{k}_j)$ for $j \in \{6, 7, 10\}$.

Expanding the Tree. The nodes are expanded according to their preorder index. We call the first not yet expanded node the *frontier*. In our example of GGM_4, the initial frontier is node 7, which is then expanded into

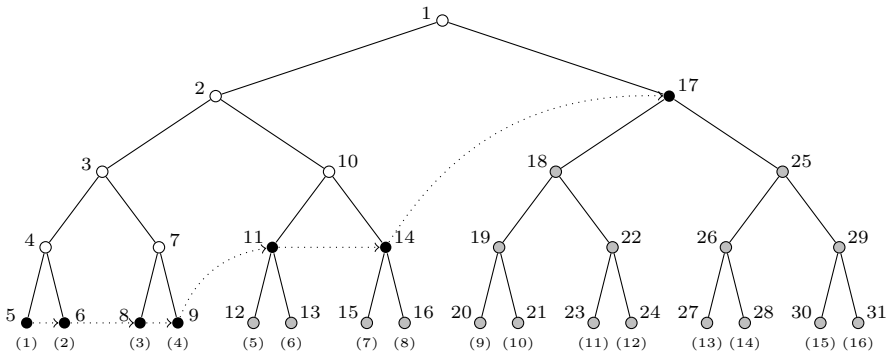

Fig. 5: A visualization of the node expansion in the enhanced GGM construction.

nodes 8 and 9. In this step, Update "replaces" (they remain on the bulletin board but are no longer needed for this party) the links $(6, 7)$ and $(7, 10)$ with the following ones: $(6, 8)$, $(8, 9)$, and $(9, 10)$. (All those newly added encryptions satisfy Invariant 1.) The new frontier is now 10. When later expanding node 10 into nodes 11 and 14, we upload links $(9, 11)$, $(11, 14)$, and $(14, 17)$, replacing the existing links $(9, 10)$ and $(10, 17)$. See Fig. 5 for the state of the tree after the expansion of $\mathsf{f} = 10$. More generally, when expanding f, we consider the following two additional nodes

- $\mathsf{f}^- := \mathsf{prevLeaf}(\mathsf{f})$ denoting the the largest leaf index $\mathsf{f}^- < \mathsf{f}$ that is not a descendent of f.
- $\mathsf{f}^+ := \mathsf{rCoPath}(\mathsf{f})$ denoting the first node on f's right copath,

and replace the links $(\mathsf{f}^-, \mathsf{f})$ and $(\mathsf{f}, \mathsf{f}^+)$ by link

- $(\mathsf{f}^-, \mathrm{leftChild}(\mathsf{f}))$,
- $(\mathrm{leftChild}(\mathsf{f}), \mathrm{rightChild}(\mathsf{f}))$,
- $(\mathrm{rightChild}(\mathsf{f}), \mathsf{f}^+)$.

We observe that by definition $\mathsf{rCoPath}(\mathrm{rightChild}(\mathsf{f})) = \mathsf{rCoPath}(\mathsf{f}) = \mathsf{f}^+$ and $\mathsf{rCoPath}(\mathrm{leftChild}(\mathsf{f})) = \mathrm{rightChild}(\mathsf{f})$. Thus, storing those additional links maintains the first of the following invariant that will become crucial for fast forwarding.

I2. For any v not on the leftmost path, if s_v has been computed, then the link $(v, \mathsf{rCoPath}(v))$, i.e., the ciphertext $\mathsf{SE.Enc}(\mathsf{k}_v, (\mathsf{s}_{\mathsf{rCoPath}(v)}, \mathsf{k}_{\mathsf{rCoPath}(v)}), \mathsf{rCoPath}(v))$, has been added to the bulletin board.
I3. For any leaf v except the leftmost one, if s_v has been computed, then the link $(\mathsf{prevLeaf}(v), v)$ has been added to the bulletin board.

To be able to efficiently create those links described above, our algorithm keeps at any point in time the index, seed, and key of the f, f^-, and f^+ as part of the state. After the expansion, those pointers of course have to be adjusted and the respective seeds and keys locally stored.

- If $\mathrm{leftChild}(\mathsf{f})$ is not a leaf, then f' becomes this node. The new f^+ is thus is the right sibling that we also just derived and f^- remains unchanged.
- If $\mathrm{leftChild}(\mathsf{f})$ is a leaf, then this node becomes the new f^-. The new f gets the old f^+. Its first right copath node becomes the new f^+. While for the latter we do not have seed or key readily stored, we know from Invariant 2 that there is a link from the old to the new f^+ stored in the bulletin board that we can use to retrieve those values.

Sequential Updates. For each sequential update from epoch e to $e + 1$, the Update algorithm has to output the seed of the $(e + 1)$-th leaf, which we denote by $\mathsf{leaf}(e + 1)$. For this to be done in a constant number of cryptographic operations, the algorithm relies on a link $(\mathsf{leaf}(e), \mathsf{leaf}(e + 1))$ is readily stored in the bulletin board. Recall from Invariant 3 that such a link exists as long as the seed $\mathsf{leaf}(e + 1)$ has been derived at this point, meaning that Update just needs to expand the frontier sufficiently fast.

We achieve this by doing two expansions per invocation of Update, as long as there are still nodes to expand. Consider the inner nodes on the copath of the leftmost leaf. Those node root $\mathsf{h} - 1$ trees $T_1, \ldots T_{\mathsf{h}-1}$ of increasing

height that still need to be expanded after Init, as shown in Fig. 4. During the first update, i.e., when moving to node 6 in our example of GGM_4, we can expand T_1. More generally, we observe that T_j has 2^j leaves and T_{j+1} has 2^j inner nodes that need to be expanded. Hence, doing two expansion steps per Update invocation maintains the following invariant:

I4. By the time the epoch advances from T_j to T_{j+1}, i.e., when transitioning from epoch e to $e+1$ such that $\mathsf{leaf}(e) \in T_j$ and $\mathsf{leaf}(e+1) \in T_{j+1}$, the tree T_{j+1} has already been fully expanded.

In summary, our algorithm achieves sequential updating using at most three elements from the bulletin board (one to derive the new epoch's output and two for the tree expansion) and two PRG expansion and uploading at most six new elements to the bulletin board.

Fast Forwarding. We now describe the process of forwarding from epoch e to $e' \gg e$ in logarithmic time. Observe that by Invariant 2 there is an encryption chain along the right copath of $\mathsf{leaf}(e)$ stored in the bulletin board. (This holds as the second node on the right copath of $\mathsf{leaf}(e)$ is equal to $\mathsf{rCoPath}(\mathsf{rCoPath}(\mathsf{leaf}(e)))$ and so forth.) Thus, Update can work analogously to the basic GGM-PRNG construction by determining the first node of this copath intersecting with $\mathsf{leaf}(e')$'s path and recover this node decrypting a logarithmic number of ciphertexts. Then, the seed and key of $\mathsf{leaf}(e')$ can be derived using a logarithmic number of PRG.Expand calls. The local state consisting of the keys and seeds of f^-, f, and f^+ can be restored analogously.

Finally observe that for the party to be able to continue from epoch e' it may not sufficient to just recover the local state, as subsequent calls to both Update and Leap require certain links to be stored in the bulletin board. For our setting, where we assume that Leap is only used to catch up with other parties, this is not an issue, however. For each epoch between e and e' the first party reaching it must have done so using a sequential update, uploading all necessary encryptions as part of this process.

Efficiency. Let n denote the maximal number of epochs, i.e., $n = 2^{\mathsf{h}}$. Then, the Init algorithm performs $\mathcal{O}(\log(n))$ many cryptographic operations and uploads this many elements to the bulletin board. Afterwards, Update requires at most $3 = \mathcal{O}(1)$ elements from the bulletin board and performs $\mathcal{O}(1)$ cryptographic operations and uploads at most $6 = \mathcal{O}(1)$ elements. So far we have glossed over how the $\mathsf{UpdateSK\text{-}Idx}(e)$ algorithm works. In short, it needs to be able to compute $\mathsf{leaf}(e)$, the f corresponding to $\mathsf{leaf}(e)$, $\mathsf{rCoPath}(\mathsf{f})$ and $\mathsf{prevLeaf}(\mathsf{f})$. Each of them can be easily computed in time $\mathcal{O}(\log n)$ given that f advances at the predictable double speed compared to $\mathsf{leaf}(e)$. Finally, Leap requires $\mathcal{O}(\log n)$ elements from the bulletin board and performs $\mathcal{O}(\log n)$ computation. In addition, $\mathsf{Leap\text{-}Idx}(e, e')$ needs to compute the elements of the right copath of $\mathsf{leaf}(e)$ that are ancestors of $\mathsf{leaf}(e')$, the corresponding frontier f', and $\mathsf{prevLeaf}(\mathsf{f}')$. It then outputs the corresponding paths to recover e', $\mathsf{prevLeaf}(\mathsf{f}')$, f', and $\mathsf{rCoPath}(\mathsf{f}')$, where the latter can be directly recovered from f'. All of those computations can be done in $\mathcal{O}(\log n)$ as well.

Correctness and Security. Let us briefly summarize the main results of this section, which is that our modifications to the forward-secure GGM-based PRNG do not affect either correctness or security.

Theorem 1. *The scheme outlined in Section 4.1 is correct and secure FF-PRNG, for a bounded number of at most 2^{h} epochs, according to Definitions 3 and 4, respectively.*

Proof. We have argued correctness throughout the section. In particular, sequential updating works due to Invariants 3 and 4. For the fast-forwarding, we observed in Invariants 2 and 4 that at epoch i there is an encryption chain alongside the right copath of $\mathsf{leaf}(i)$ stored in the bulletin board, from which Leap can recover the seed of any future epoch $j > i$ in the same manner as the underlying GGM construction. Moreover, from Invariant 4 it follows that $\mathsf{f}_i^- > \mathsf{leaf}(i)$ for any epoch i. Using $\mathsf{f}_j^+ > \mathsf{f}_j > \mathsf{f}_j^- \geq \mathsf{f}_i^-$ we thus know that Leap can restore the required local analogously.

Finally, the same observation means that the local state in epoch i only contains seeds for epochs strictly greater than i, and keys for epochs larger or equal to i. Also k_i reveals no information about s_i. Combined with Invariant 1 this implies that the local state together with the content of the bulletin board only leaks information about nodes $j > i$ (assuming the encryption scheme to be IND-CPA secure). Hence, the security directly reduces directly to the security of the basic punctured GGM construction. □

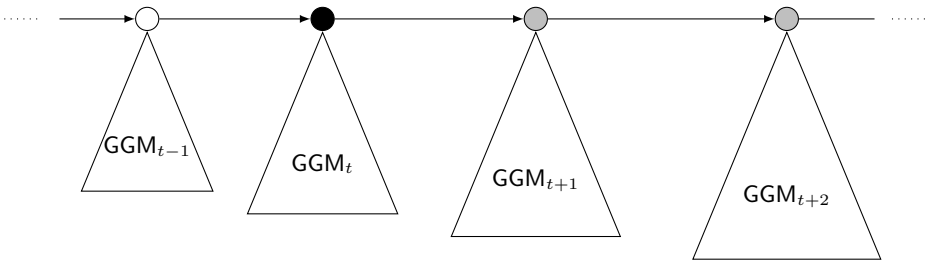

Fig. 6: The sequence of GGM trees of increasing height underlying the unbounded scheme. The current epoch is part of the tree GGM_t, whereas expansion either happens in GGM_t or GGM_{t+1}.

4.2 Supporting an Unbounded Number of Epochs

In this section, we now briefly outline how our construction can be extended to support an unbounded number of epochs, and in the process reduce the running time of Init to $\mathcal{O}(1)$.

In a nutshell, we can apply the idea of a sequence of GGM trees of growing height, as used in [42]. Their roots can be derived using a forward-secure PRNG, such as the folklore construction from PRG.Expand. Recall from Invariant 4 that within a tree GGM_t, Update is done expanding before the epoch reaches the subtree T_{t-1} (cf. Fig. 4). As this subtree has 2^{t-1} more leaves, we can spend this time initializing the next tree GGM_{t+1} instead, deriving its leftmost path and storing encryptions of its copath on the bulletin board.

In a bit more detail the scheme works as follows. See Fig. 6 for a graphical depiction of the strategy.

1. Init samples $\mathsf{st}_0 \in \{0,1\}^\kappa$ and computes $(\mathsf{st}_1, \mathsf{root}_1, \cdot, \cdot) \leftarrow \mathsf{PRG.Expand}(\mathsf{st}_0)$, and runs Init of the fixed-height scheme from the previous section with GGM_1. The state consists of st_1, the index of the current epoch's tree $t = 1$, and the state of the underlying construction.
2. Update runs the corresponding algorithm of the underlying scheme on the tree GGM_t to produce the output R_{e+1} for the new epoch $e+1$. Moreover, if there are still nodes to expand in GGM_t, then we use the same strategy as in the underlying scheme. Otherwise, if GGM_{t-1} has not been fully initialized, the scheme derives $(\mathsf{st}_{t+1}, \mathsf{root}_{t+1}, \cdot, \cdot) \leftarrow \mathsf{PRG.Expand}(\mathsf{st}_t)$ if needs be, and then performs one more step of the initialization. If at the end of Update the tree GGM_t has been exhausted, i.e., all leaves have been output, then we delete st_t and increments t.
3. If e' lies in the same tree as e, then derive the output the same way as the basic scheme. Otherwise, Leap first derives the root of the new epoch e', and then the corresponding leaf. The state around the current frontier is derived analogously.

Theorem 2. *The FF-PRNG outlined in Section 4.2 is correct and secure according to Definitions 3 and 4, respectively. Moreover,* Init *and* Update *both perform* $\mathcal{O}(1)$ *cryptographic operations and upload* $\mathcal{O}(1)$ *elements to the bulletin board, and* Update-Idx *outputs an index set of size* $\mathcal{O}(1)$*. Finally,* Leap *requires* $\mathcal{O}(\log j)$ *elements from the bulletin board and has running time in* $\mathcal{O}(\log j)$*.*

Proof. For correctness it is sufficient to observe that whenever the epoch moves to a tree GGM_t, the initialization of this tree has been completed, as this needs at most $t + 1 \leq |T_{t-2}| = 2^{t-3}$ operations. Security also follows directly from the bounded-epochs scheme and the observation that st_t does not leak any information about the trees GGM_s for $s \leq t$. For efficiency, we observe that an epoch i is at most in the tree $t \leq \log_2(i)$, due to the increasing height. Hence, when fast-forwarding to an epoch j, we need to derive at most $\mathcal{O}(t)$ roots and then, analogous to the basic scheme, perform $\mathcal{O}(t)$ computation there. Finally, initializing GGM_1 is clearly a $\mathcal{O}(1)$ operation. □

5 Fast-Forwardable Updatable Public-Key Encryption

Our generic FF-UPKE uses any update-homomorphic UPKE (H-UPKE) and is built around the idea of so-called *cumulative updates*, i.e., update ciphertexts that aggregate a sequence of individual updates. We use an *update graph* to govern which cumulative updates are produced, to balance the senders' overhead with the receiver's ability to fast forward. (For instance, the complete update graph would allow the receiver to update in constant time while imposing an undesirable linear overhead on each sender, while the empty update graph results in a plain UPKE without fast-forwarding.)

5.1 Update-Homomorphic UPKE

As a building block — to allow for cumulative updates — our construction makes use of an update-homomorphic UPKE scheme, constituting a special case of updatable public-key encryption.

In brief, in addition to a key-generation algorithm $(\mathsf{pk}_1, \mathsf{sk}_1, \mathsf{pp}) \leftarrow \mathsf{KeyGen}(1^\kappa)$, and respective message encryption and decryption algorithms $\mathsf{c} \leftarrow \mathsf{Encrypt}(\mathsf{pk}_i, \mathsf{m})$ and $\mathsf{m} \leftarrow \mathsf{Decrypt}(\mathsf{sk}_i, \mathsf{c})$, an update-homomorphic UPKE scheme provides the following structure:

(1) update ciphertext consist of an encrypted update message, i.e., $\mathsf{up}_{i+1} \leftarrow \mathsf{UpdEnc}(\mathsf{pk}_i, \delta_{i+1})$, sampled using $\delta_{i+1} \leftarrow \mathsf{UpdGen}(\mathsf{pp})$ based on the public parameters pp;
(2) update messages are elements from the secret-key space which forms a group under some operator $\star$;
(3) the secret keys are updated according to $\mathsf{sk}_{i+1} = \mathsf{sk}_i \star \delta_{i+1}$;
(4) UpdEnc is message homomorphic, i.e., there is an algorithm $\mathsf{Upd\text{-}Comb}(\mathsf{up}, \mathsf{up}')$ homomorphically combining two updates encrypted under the same public key pk_i.

Using the shorthand notation $\Delta_{[j,\ell]} \coloneqq (\delta_{j+1} \star \cdots \star \delta_\ell)$, the homomorphism property thus ensures that we can compute an encryption that is equivalent to $\mathsf{Up}^i_{[j,\ell]} \leftarrow \mathsf{UpdEnc}(\mathsf{pk}_i, \Delta_{[j,\ell]})$ from two partial updates $\mathsf{Up}^i_{[j,k]}$ and $\mathsf{Up}^i_{[k,\ell]}$, for any $j < k < \ell$. More formally, we define update-homomorphic UPKE schemes as follows.

Definition 8. *An* update-homomorphic UPKE (H-UPKE) scheme *is a tuple of algorithms* $(\mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Decrypt}, \mathsf{UpdGen}, \mathsf{UpdEnc}, \mathsf{UpdDec}, \mathsf{UpdatePK}, \mathsf{Upd\text{-}Comb})$ *for which the secret-key space $\mathcal{SK}$ forms a semigroup (i.e., is associative with respect to some operator $\star$) and the algorithms are defined as follows:*

- *the* key-generation *algorithm* $(\mathsf{pk}_1, \mathsf{sk}_1, \mathsf{pp}) \leftarrow \mathsf{KeyGen}(1^\kappa)$*, which outputs an initial key pair* sk_1 *and* pk_1*, as well as public parameters* pp*;*
- *the* encryption *algorithm* $\mathsf{c} \leftarrow \mathsf{Encrypt}(\mathsf{pk}_i, \mathsf{m})$ *and the and deterministic* decryption *algorithm* $\mathsf{m} \leftarrow \mathsf{Decrypt}(\mathsf{sk}_i, \mathsf{c})$*, respectively;*
- *the* update-sample *algorithm* $\delta_{i+1} \leftarrow \mathsf{UpdGen}(\mathsf{pp})$ *producing* $\delta_{i+1} \in \mathcal{SK}$*;*
- *the deterministic* public-key update *algorithm* $\mathsf{pk}_{i+1} \leftarrow \mathsf{UpdatePK}(\mathsf{pk}_i, \delta_{i+1})$*, which given a public key and an update message produces an updated one;*
- *the* update-encryption *algorithm* $\mathsf{up}^i_j \leftarrow \mathsf{UpdEnc}(\mathsf{pk}_i, \delta_j)$*, for $j > i$;*
- *the* update-combination *algorithm* $\mathsf{Up}^i_{[j,\ell]} \leftarrow \mathsf{Upd\text{-}Comb}(\mathsf{Up}^i_{[j,k]}, \mathsf{Up}^i_{[k,\ell]})$*, merging two updates encrypted under the same public key* pk_i*.*
- *the and deterministic* update-decryption $\Delta^i_{[j,\ell]} \leftarrow \mathsf{UpdDec}(\mathsf{sk}_i, \mathsf{Up}^i_{[j,\ell]})$*;*

Correctness and Security. We formalize correctness using two separate properties. The first property essentially demands that the pairs $(\mathsf{Encrypt}, \mathsf{Decrypt})$ and $(\mathsf{UpdEnc}, \mathsf{UpdDec})$ represent correct pairs of encryption and decryption algorithms for their respective message spaces — analogously to the standard UPKE definition. This is formalized in the game on the left side of Fig. 7. To account for the evolving sequence of public and secret keys, as well as the use of bad randomness, the game allows the adversary to update the keys an arbitrary number of periods under his randomness before submitting a challenge message to be encrypted. The adversary wins if either the ciphertext or one of the update messages gets decrypted incorrectly. The second property concerns homomorphism and is, thus, unique to update-homomorphic UPKE. It requires that the output of $\mathsf{Upd\text{-}Comb}$ must correctly decrypt to the multiplication of the underlying update secrets (for the group operator $\star$), i.e., that

$$\Delta_{[j,k]} \star \Delta_{[k,\ell]} = \mathsf{UpdDec}\Big(\mathsf{sk}_i, \mathsf{Upd\text{-}Comb}\big(\mathsf{UpdEnc}(\mathsf{pk}_i, \Delta_{[j,k]}), \mathsf{UpdEnc}(\mathsf{pk}_i, \Delta_{[k,\ell]})\big)\Big),$$

which is formalized in the game on the bottom of Fig. 7.

Definition 9 (Correctness). *A UPKE scheme is correct if any PPT adversary $\mathcal{A}$ has a negligible probability of winning either the correctness or the update-homomorphicity game the game depicted in Fig. 7.*

Finally, we require IND-CPA security. The IND-CPA game is essentially the same as for regular UPKE, when accounting for the imposed special structure of the updating mechanism, via the sender invoking

1. $\delta \leftarrow \mathsf{UpdGen}(\mathsf{pp})$

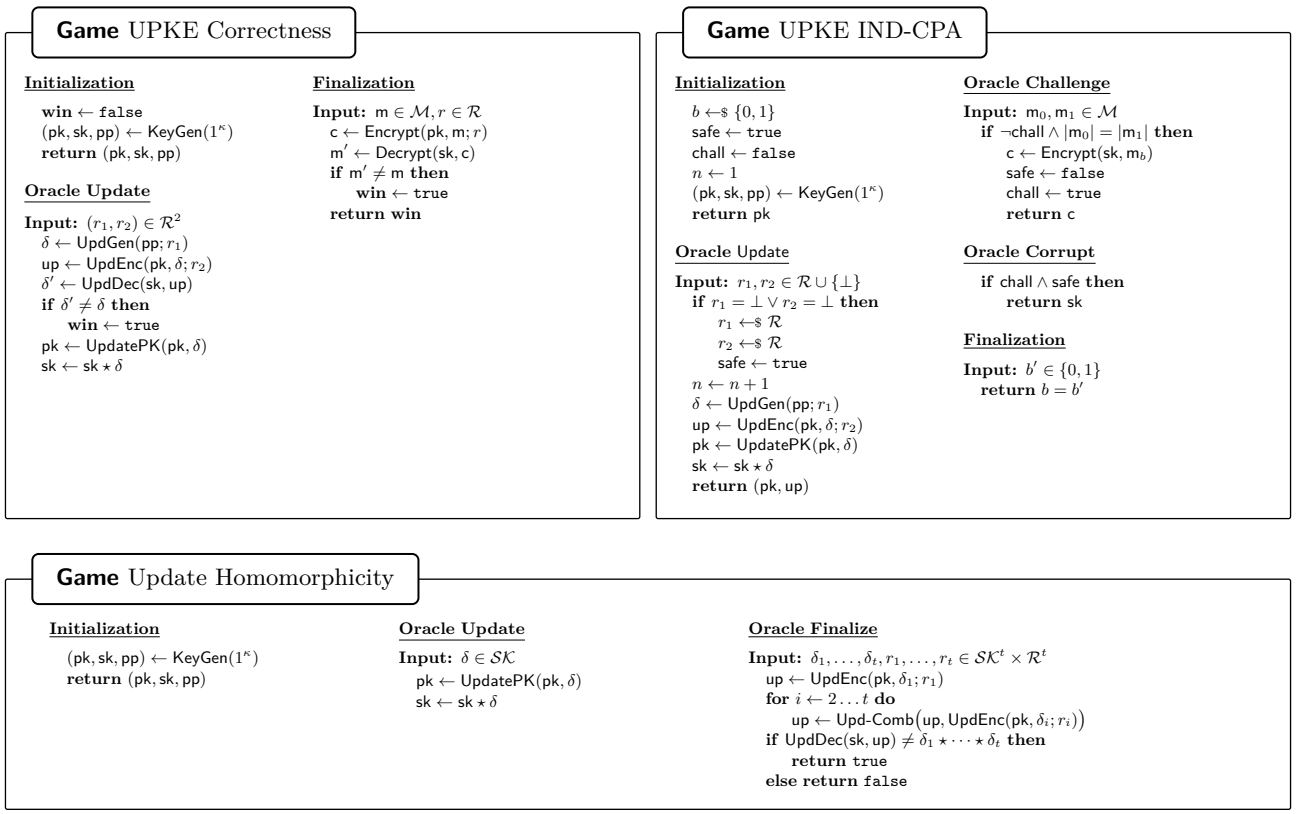

Fig. 7: The correctness (left), IND-CPA (right), and homomorphism (bottom) games of Update-Homomorphic Updatable Public-Key Encryption (H-UPKE).

2. $\mathsf{up} \leftarrow \mathsf{UpdEnc}(\mathsf{pk}, \delta)$
3. $\mathsf{pk}' \leftarrow \mathsf{UpdatePK}(\mathsf{pk}, \delta)$,

and on the other side the receiver using $\delta \leftarrow \mathsf{UpdDec}(\mathsf{sk}, \mathsf{up})$ and $\mathsf{sk}' \leftarrow \mathsf{sk} \star \delta$. A formal description of the resulting IND-CPA game can be found on the right hand side of Fig. 7.

Definition 10 (Security). *A UPKE scheme is said to be IND-CPA secure if any PPT adversary $\mathcal{A}$ has a negligible probability of winning the game depicted in Fig. 7.*

Schemes. In Appendix B we show that with minor modifications the standard-model UPKE schemes introduced in the recent work of Dodis et al. [24] do lend themselves to an update-homomorphic UPKE scheme, resulting in instantiations under either the DDH or the LWE assumption. We remark that both constructions have some (different) caveats: the LWE-based scheme supports a bounded number of homomorphic operations, supporting aggregation of at most q atomic updates (but q can be chosen superpolynomially large at the expense of slightly larger other parameters) while the DDH-based construction supports an a priori unbounded number of aggregations but decryption of an aggregated update takes local computation time $\mathcal{O}(\sqrt{n})$ in the number of underlying updates n.

5.2 Update Graphs

A crucial part of our construction will be deciding which cumulative updates to generate. If, on the one hand, we insert too few such cumulative updates, then $\mathsf{LeapSK\text{-}Idx}(i, j) \approx j - i$ loses the fast-forward property. If, on the other hand, we insert too many — e.g., all of them — then both $\mathsf{UpdatePK}$ will need both to read and write linearly many elements from the bulletin board. Indeed, such a solution would represent in many aspects the

trivial dual solution to no fast-forwarding, as the former requires linear bandwidth for the receiver whereas the latter requires linear bandwidth for all the senders.

To examine those trade-offs in more detail, we reformulate the insertion of cumulative updates as a graph-theoretic problem. To simplify the reasoning about the index set, we moreover include the atomic (non fast-forward) updates into the graph, as formalized by the following definition.

Definition 11. *Let $\alpha, \beta, \gamma \colon \mathbb{N} \to \mathbb{N}$. An (α, β, γ)-*update graph $\mathcal{G} = (\mathbb{N}, E)$ *is a directed acyclic graph with the following properties:*

1. $\forall i \in \mathbb{N} : (i, i+1) \in E$,
2. $\forall (i,j) \in E : i < j$,
3. $\forall n \in \mathbb{N} : \deg^{\mathsf{in}}_{\mathcal{G}}(n) \leq \alpha(n)$,
4. $\forall n \in \mathbb{N} : \mathrm{diam}(\mathcal{G}_n) \leq \beta(n)$,
5. $\forall n \in \mathbb{N} : |\mathrm{active}_{\mathcal{G}}(n)| \leq \gamma(n)$,

where $\mathrm{active}_{\mathcal{G}}(n) \coloneqq \{i \in [n-1] \mid \exists j > n : (i,j) \in E\}$, *and* $(\mathcal{G}_n)_{n \in \mathbb{N}}$ *with* $\mathcal{G}_n \coloneqq ([n], E_n)$ *and* $E_n \coloneqq E \cap [n]^2$ *denotes the sequence of prefix graphs.*

Looking slightly ahead, let us briefly consider how the different parameters will affect the efficiency of our construction. First, the number of update messages required by LeapSK is bounded by $\beta(j)$. Second, $\mathrm{active}_{\mathcal{G}(n)} \leq \gamma(n)$ corresponds to the set of cumulative updates that need to be extended when a sender initiates the i-th epoch using $\mathsf{UpdatePK}$. Finally, the indegree represents the number of "finalized" updates for the respective epoch. This mainly becomes of relevance if the FF-UPKE scheme is deployed in a single-sender setting.

To be of use for our construction, we need that a given update graph can be efficiently computed. Specifically, our construction will need to compute the sets $E^{\mathsf{in}}_{\mathcal{G}}(i)$ and $\mathrm{active}_{\mathcal{G}}(i)$ for each node i, as well as computing short paths between any nodes i and j.

Definition 12. *We say that an (α, β, γ)-update graph $\mathcal{G} = (\mathbb{N}, E)$ is implemented by a pair of deterministic algorithms $(\mathcal{G}.\mathsf{Eval}, \mathcal{G}.\mathsf{Path})$ if:*

- $\mathcal{G}.\mathsf{Eval}(n)$ *outputs* $E^{\mathsf{in}}(n)$ *and* $\mathrm{active}(n)$ *in* $\mathcal{O}(\mathsf{poly}(\log n))$ *time;*
- $\mathcal{G}.\mathsf{Path}(i,j)$ *outputs a path* $\vec{e}$ *from node* i *to* j *such that* $|\vec{e}| \leq \beta(j)$ *in* $\mathcal{O}(\mathsf{poly}(\beta(j) \cdot j))$.

5.3 A Generic Construction

We now construct a Fast-Forwardable UPKE scheme based on an Update-Homomorphic UPKE scheme and an update graph. The basic idea is very simple: When the sender j chooses the corresponding update secret δ_{j+1}, in addition to updating pk_j to pk_{j+1}, they will also

(1) produce fresh ciphertext $\mathsf{Up}_{[j,j+1]}$ encrypting δ_{j+1} under pk_j; and
(2) for every $i \in \mathrm{active}(j+1) \cup E^{\mathsf{in}}(j+1)$, fetch $\mathsf{Up}_{[i,j]}$ from the bulletin board, and use the update-homomorphic property of the UPKE to compute ciphertexts $\mathsf{Up}_{[i,j+1]}$ to be published in the bulletin board.

On the receiving side, if the receiver knows key sk_i, and wishes to jump to some key sk_j for $j > i$, it will:

(1) compute a short "leap path" $i = i_0 \to i_i \to \cdots \to i_d = j$ in the update graph;
(2) retrieve d ciphertexts $\{\mathsf{Up}_{[i_k, i_{k+1}]}\}$ from the bulletin board;
(3) decrypt Up_{i_0,i_1} using $\mathsf{sk}_i = \mathsf{sk}_{i_0}$ to get $\Delta_{[i_0,i_1]}$;
(4) compute $\mathsf{sk}_{i_1} \coloneqq \mathsf{sk}_{i_0} \star \Delta_{[i_0,i_1]}$;
(5) iterate steps (3)-(4) d times to finally "catch up" with $\mathsf{sk}_j = \mathsf{sk}_{i_d}$.

A formal description of the scheme is presented in Fig. 8.

Protocol Fast-Forwardable UPKE

KeyGen(1^κ)

$\mathsf{BB}_{init}[\cdot] \leftarrow \bot$
$(\mathsf{pk}_{\text{H-UPKE}}, \mathsf{sk}_{\text{H-UPKE}}, \mathsf{pp}_{\text{H-UPKE}}) \leftarrow \text{H-UPKE.KeyGen}(1^\kappa)$
$\mathsf{pk} \leftarrow (1, \mathsf{pk}_{\text{H-UPKE}}, \mathsf{pp}_{\text{H-UPKE}})$
$\mathsf{BB}[\cdot,\cdot] \leftarrow \bot$
$\mathsf{pk} \leftarrow (1, \mathsf{pk}_{\text{H-UPKE}}, \mathsf{pp}_{\text{H-UPKE}}, \mathsf{BB})$
$\mathsf{sk} \leftarrow (1, \mathsf{sk}_{\text{H-UPKE}})$
return $(\mathsf{pk}, \mathsf{sk}, \mathsf{BB}_{init})$

Encrypt(pk, m)

parse $(\cdot, \mathsf{pk}_{\text{H-UPKE}}, \cdot) \leftarrow \mathsf{pk}$
parse $(\cdot, \mathsf{pk}_{\text{H-UPKE}}, \cdot, \cdot) \leftarrow \mathsf{pk}$
$\mathsf{c} \leftarrow \text{H-UPKE.Encrypt}(\mathsf{pk}_{\text{H-UPKE}}, \mathsf{m})$
return c

Decrypt(sk, c)

parse $(\cdot, \mathsf{sk}_{\text{H-UPKE}}) \leftarrow \mathsf{sk}$
$\mathsf{m} \leftarrow \text{H-UPKE.Encrypt}(\mathsf{sk}_{\text{H-UPKE}}, \mathsf{c})$
return m

UpdatePK(pk, BB)

parse $(n, \mathsf{pk}_{\text{H-UPKE}}, \mathsf{pp}_{\text{H-UPKE}}) \leftarrow \mathsf{pk}$
parse $(n, \mathsf{pk}_{\text{H-UPKE}}, \mathsf{pp}_{\text{H-UPKE}}, \mathsf{BB}) \leftarrow \mathsf{pk}$
$n \leftarrow n + 1$
$\mathsf{BB}'[\cdot,\cdot] \leftarrow \bot$

// Generate the regular update
$\delta \leftarrow \text{H-UPKE.UpdGen}(\mathsf{pp})$
$\mathsf{pk}'_{\text{H-UPKE}} \leftarrow \text{H-UPKE.UpdatePK}(\mathsf{pk}, \delta)$
$\mathsf{up}_{(n-1,n)} \leftarrow \text{H-UPKE.UpdEnc}(\mathsf{pk}, \delta)$

// Update all active cumulative updates
$(\mathsf{In}, \mathsf{A}) \leftarrow \mathcal{G}.\mathsf{Eval}(n)$
for all $i \in \mathsf{A} \cup \mathsf{In} \setminus \{n-1\}$ **do**
  $\mathsf{up}_{(i,n-1)} \leftarrow \mathsf{BB}[i, n-1]$
  $\mathsf{up}_{(i,n)} \leftarrow \text{H-UPKE.Upd-Comb}(\mathsf{up}_{(i,n-1)}, \mathsf{up}_{(n-1,n)})$
  $\mathsf{BB}'[i, n] \leftarrow \mathsf{up}_{(i,n)}$
$\mathsf{BB}'[n-1, n] \leftarrow \mathsf{up}_{(n-1,n)}$
$\mathsf{pk}' \leftarrow (n, \mathsf{pk}'_{\text{H-UPKE}}, \mathsf{pp}_{\text{H-UPKE}})$
return $(\mathsf{pk}', \mathsf{BB}')$
$\mathsf{BB}''[\cdot,\cdot] \leftarrow \bot$
$\mathsf{BB}''[n-1, n] \leftarrow \mathsf{up}_{(n-1,n)}$
$\mathsf{pk}' \leftarrow (n, \mathsf{pk}'_{\text{H-UPKE}}, \mathsf{pp}_{\text{H-UPKE}}, \mathsf{BB}')$
return $(\mathsf{pk}', \mathsf{BB}'')$

UpdateSK(sk, BB)

parse $(n, \mathsf{sk}_{\text{H-UPKE}}) \leftarrow \mathsf{sk}$
$n \leftarrow n + 1$
$\delta \leftarrow \text{H-UPKE.UpdDec}(\mathsf{sk}_{\text{H-UPKE}}, \mathsf{BB}[n-1, n])$
$\mathsf{sk}_{\text{H-UPKE}} \leftarrow \mathsf{sk}_{\text{H-UPKE}} \star \delta$
$\mathsf{sk}' \leftarrow (n, \mathsf{sk}_{\text{H-UPKE}})$
return sk'

LeapSK(sk, j, BB)

parse $(n, \mathsf{sk}_{\text{H-UPKE}}) \leftarrow \mathsf{sk}$
if $j \leq n$ **then**
  return $\bot$
$\vec{e} \leftarrow \mathcal{G}.\mathsf{Path}(n, j)$
for $(u, v) \in \vec{e}$ **do**
  $\delta \leftarrow \text{H-UPKE.UpdDec}(\mathsf{sk}_{\text{H-UPKE}}, \mathsf{BB}[u, v])$
  $\mathsf{sk}_{\text{H-UPKE}} \leftarrow \mathsf{sk}_{\text{H-UPKE}} \star \delta$
$n \leftarrow j$
$\mathsf{sk}' \leftarrow (n, \mathsf{sk}'_{\text{H-UPKE}})$
return sk'

UpdatePK-Idx(n)

$\mathrm{I} \leftarrow \varnothing$
$(\mathsf{In}, \mathsf{A}) \leftarrow \mathcal{G}.\mathsf{Eval}(n+1)$
for all $i \in \mathsf{A} \cup \mathsf{In} \setminus \{n\}$ **do**
  $\mathrm{I} \leftarrow \mathrm{I} \cup \{(i, n)\}$
return I

UpdateSK-Idx(n)

return $\{(n, n+1)\}$

LeapSK-Idx(n, j)

$\vec{e} \leftarrow \mathcal{G}.\mathsf{Path}(n, j)$
$\mathrm{I} \leftarrow \varnothing$
for $(u, v) \in \vec{e}$ **do**
  $\mathrm{I} \leftarrow \mathrm{I} \cup \{(u, v)\}$
return I

Fig. 8: The FF-UPKE protocols are built from an Update-Homomorphic UPKE scheme H-UPKE and an update graph $\mathcal{G}$. Lines enclosed in solid boxes belong to the regular multi-sender protocol, whereas lines enclosed in dashed boxes represent the single-sender variant.

A Single-Sender Variant. When deployed in a single-sender setting, such as a two-party secure messaging scheme (considered as the original motivation for UPKE by Jost et al. [38]) the scheme can be slightly tweaked for a different storage/bandwidth trade-off. To this end, we observe that in our scheme the receiver will never access the temporary "ongoing" update messages but only ever use an element (i, j) if $(i, j) \in E$. As a consequence, the sender may choose to upload only the elements from $E^{\mathsf{in}}(n)$ while keeping the ongoing cumulative updates as part of his local state.

This works nicely since the sender does not need arbitrary $\gamma(n) + \alpha(n)$ many of the $\mathcal{O}(n)$ so far uploaded elements, but in each step we have

$$\text{active}(n) \cup E^{\mathsf{in}}(n) \subseteq \text{active}(n-1) \cup \{n-1\},$$

implying that $\mathcal{O}(\gamma + \alpha(n))$ storage suffices. Additionally, this variant has the distinct advantage that $\mathsf{UpdatePK}$ does not need to read anything from the bulletin board, i.e., $\mathsf{UpdatePK\text{-}Idx}(n) = \varnothing$, potentially reducing latency.

The corresponding protocol is depicted in Fig. 8 as well using the dashed boxes. For simplicity, we model the local state as the public key containing a "local" bulletin board.

Correctness and Efficiency. Correctness of the construction follows essentially directly from the correctness of the underlying Update-Homomorphic UPKE scheme, which is formalized in parts of the correctness of a regular UPKE scheme plus the correctness condition of the homomorphism. Moreover, by the construction the various parameters of the update graph directly translate to the efficiency of the scheme, yielding the following result.

Theorem 3. *The FF-UPKE schemes presented in Fig. 8 are correct, according to Definition 6, if the underlying scheme* $\mathsf{H\text{-}UPKE}$ *is correct and update homomorphic as formalized via the games from Fig. 7.*

Moreover, the regular scheme has public and secret keys of roughly the same size, and encryption and decryption of the same efficiency, as the underlying $\mathsf{H\text{-}UPKE}$ *scheme. Using an* (α, β, γ)*-update graph, yields the following efficiency:*

- $\mathsf{UpdatePK}$*:* $|\mathsf{UpdatePK\text{-}Idx}(n)| \leq \gamma(n+1)$ *and* $|\mathsf{BB}_{up}| \leq \gamma(n+1)+1$*. Moreover,* $\mathsf{UpdatePK}$ *needs* $\mathcal{O}(\gamma(n+1))$ *as many cryptographic operations as the underlying scheme.*
- $\mathsf{UpdateSK}$*:* $|\mathsf{UpdateSK\text{-}Idx}(n)| = 1$ *and* $\mathsf{UpdateSK}$ *uses* $\mathcal{O}(1)$ *as many cryptographic operations as the underlying scheme.*
- LeapSK*:* $|\mathsf{LeapSK\text{-}Idx}(n, j)| \leq \beta(j)$ *and* LeapSK *uses* $\mathcal{O}(\beta(j))$ *as many cryptographic operations as the underlying scheme.*

The single-sender scheme has a secret key of roughly the same size as the underlying $\mathsf{H\text{-}UPKE}$ *scheme, and a public key size of roughly* $\gamma(n)$ *as big as the underlying scheme (modeling local storage) and the same efficiency except that* $|\mathsf{UpdatePK\text{-}Idx}(n)| = 0$ *and* $|\mathsf{BB}_{up}| \leq \alpha(n+1)$*.*

Proof. The efficiency claims follow immediately from the construction. For correctness, we first observe that as long as the receiver updates sequentially, i.e., use $\mathsf{UpdateSK}$, then both variants essentially execute the underlying update-homomorphic UPKE scheme. Hence, the decryption correctness property of Definition 6 follows from the correctness of the underlying scheme.

Finally, consider the fast-forward correctness, i.e., that $\mathsf{LeapSK}(\mathsf{st}_i, j)$ results in the same secret key as sequentially updating. Here we first observe that the receiver only accesses $\mathsf{BB}[i, j]$ if (i, j) is an edge in the update graph and both variants upload a corresponding cumulative update when transitioning to epoch j as $i \in E^{\mathsf{in}}(j)$. Next, note that $\text{active}(n) \cup E^{\mathsf{in}}(n) \subseteq \text{active}(n-1) \cup \{n-1\}$ moreover ensures that whenever a sender tries to extend a cumulative update, he can find it on the bulletin board. The correctness of those cumulative updates follows from the update homomorphicity property (as formalized in Fig. 7). Hence, every update $(u, v) \in \mathcal{G}.\mathsf{Path}(i, j)$ the receiver applies during LeapSK correctly results in sk_v, finally resulting in sk_j after applying the whole path. □

Corollary 1. *When instantiated using the* $(2, \mathcal{O}(\log n), \mathcal{O}(\log n))$*-update graph construction from Theorem 8 (cf. Section 6), and either the LWE- or DDH-based update-homomorphic UPKE scheme from Theorem 9 or Theorem 10, respectively, the FF-UPKE scheme from Fig. 3 has the following communication complexity:*

- $\mathsf{UpdatePK}$ *to epoch* i *has communication complexity* $\mathcal{O}(\log(i) \cdot \kappa^3)$ *in the multi-sender variant and* $\mathcal{O}(\kappa^3)$ *in the single-sender variant;*
- $\mathsf{UpdateSK}$ *to epoch* i *has communication complexity* $\mathcal{O}(\kappa^3)$*, independent of the epoch* i*;*
- LeapSK *from epoch* i *to* j *has communication complexity* $\mathcal{O}(\log(j)\kappa^3)$*.*

Security. Security also follows directly from the security of the underlying Update-Homomorphic UPKE scheme, as intuitively the cumulative updates represent a computation on public data.

Theorem 4. *The FF-UPKE schemes presented in Fig. 8 are IND-CPA secure, according to Definition 7, if the underlying scheme* H-UPKE *is IND-CPA secure according to Definition 10.*

Proof. Security-wise it is irrelevant whether certain data is stored as part of the bulletin board or the public key, we thus focus on the regular scheme (not the single-sender variant) only.

We now show the reduction that transforms an adversary $\mathcal{A}$ against the FF-UPKE security game into an adversary $\mathcal{A}'$ against the H-UPKE security of the underlying scheme. The reduction internally runs $\mathcal{A}$ and emulates the FF-UPKE security game towards $\mathcal{A}$. It internally maintains the current bulletin board and current public key. It works as follows:

- It takes the initial public key $\mathsf{pk}_{\mathsf{H\text{-}UPKE}}$ and the public parameters $\mathsf{pp}_{\mathsf{H\text{-}UPKE}}$ output by the initialization of the H-UPKE game, and hands $\mathsf{pk} := (1, \mathsf{pk}_{\mathsf{H\text{-}UPKE}}, \mathsf{pp}_{\mathsf{H\text{-}UPKE}})$ together with an empty bulletin board $\mathsf{BB}[\cdot] \leftarrow$ to $\mathcal{A}$.
- Upon $\mathcal{A}$ invoking the update oracle with randomness $r = (r_1, r_2)$, the reduction queries the update oracle of the H-UPKE game with the same randomness.
  1. This generates an update message δ which gets encrypted as up and gets applied as $\mathsf{pk}'_{\mathsf{H\text{-}UPKE}} = \mathsf{UpdatePK}(\mathsf{pk}'_{\mathsf{H\text{-}UPKE}}, \delta)$.
  2. We observe that to simulate the $(\mathsf{pk}, \mathsf{BB}_{up}) \leftarrow \mathsf{UpdatePK}(\mathsf{pk}, \restriction_{\mathrm{I}_{\mathsf{pk}}}; r)$ the FF-UPKE game performs, we need to update all active cumulative updates. The reduction can simply perform this using the H-UPKE.Upd-Comb algorithm and store the resulting cumulative updates in BB_{up}.
  3. Finally, to emulate the FF-UPKE game, the reduction needs to emulate the $\mathsf{UpdateSK}(\mathsf{sk}, \mathsf{BB}\restriction_{\mathrm{I}_{\mathsf{sk}}})$ invocation. To this end, observe that for the specific scheme we have $\mathrm{I}_{\mathsf{sk}} = \mathsf{UpdateSK\text{-}Idx}(n) = \{(n, n+1)\}$, implying that $\mathsf{BB}\restriction_{\mathrm{I}_{\mathsf{sk}}}$ just consists of the above update ciphertext up. Furthermore, we observe that the reduction can simply call the UpdateSK oracle of the underlying H-UPKE game to have the same desired effect on the secret key. (The reduction keeps separately track of the receiver's epoch number.)
- Upon $\mathcal{A}$ invoking the challenge oracle with messages m_0 and m_1, the reduction invokes the challenge oracle of the underlying H-UPKE game and returns the obtained ciphertext c. Note that FF-UPKE and H-UPKE games maintain the safe and chall alike, meaning that the H-UPKE game will perform the appropriate checks.
- Upon $\mathcal{A}$ invoking the corruption oracle, the reduction invokes the H-UPKE game's corruption oracle to obtain $\mathsf{sk}_{\mathsf{H\text{-}UPKE}}$. It then returns $\mathsf{sk} = (n, \mathsf{sk}_{\mathsf{H\text{-}UPKE}})$ where n denotes the receiver's epoch number that is maintained by the reduction. (If the H-UPKE game returns an error due to $\mathsf{chall} \wedge \mathsf{safe}$ check failing, this error is propagated to $\mathcal{A}$.)
- Upon $\mathcal{A}$ outputting the guess b', the reduction forwards it to the underlying H-UPKE game.

In both cases, $b = 0$ and $b = 1$, the reduction perfectly emulates the IND-CPA game of the FF-UPKE scheme with the same bit, resulting in the reduction having the same advantage as $\mathcal{A}$. □

5.4 A Remark on More Efficient Schemes

The efficiency of the aforementioned FF-UPKE construction directly hinges on the efficiency of the underlying H-UPKE scheme. Unfortunately, neither construction is practically efficient, suffering from public keys growing linearly in the security parameter and update ciphertexts growing quadratically in the security parameter. Given that neither construction has been particularly optimized (and presents more feasibility results) and UPKE is a rather new primitive, we however believe that there is plenty of room for improvement and that the FF-UPKE scheme based on update-homomorphic UPKE may very well end up more efficient than the construction based on HIBE. In the following, we remark on some roadblocks we encountered when trying to devise truly efficient schemes and outline some possible future directions.

Efficient H-UPKE schemes. While the original UPKE schemes [38,4] are essentially as efficient as (hybrid) ElGamal encryption, adapting those schemes to the update-homomorphic setting poses some seemingly inherent challenges. First, note that all of those schemes do follow the paradigm of using encrypting update secrets as update messages. None of them uses a homomorphic encryption scheme in that process, however. For instance, the original scheme introduced by Jost et al. [38] uses hashed ElGamal encryption over a group of prime order p

with UpdGen sampling a uniform $\delta \leftarrow\!\!\$\ \mathbb{Z}_p$ and encrypts it in UpdEnc. Similarly, the multiplicative variant put forward by Alwen et al. [4] draws δ uniformly from the multiplicative subgroup $\mathbb{Z}_p^*$ and uses hashed ElGamal as well.

Unfortunately, their reliance on hashed ElGamal, which destroys the homomorphicity, appears to be inherent as their respective security proofs crucially rely on the ROM as a means to break circularity. More concretely, in order for those schemes to be secure, one needs that $\mathsf{sk}_{i+1} = \mathsf{sk}_i \star \delta_{i+1}$ and $\mathsf{up}_{i+1} = \mathsf{UpdEnc}(\mathsf{pk}_i, \delta_{i+1})$ leak no information about sk_i. Intuitively, to argue this one needs, however, that up_{i+1} does not leak any information about δ_{i+1}, which in turn requires sk_i to be secure for standard ElGamal encryption creating a cycling dependence. Using the ROM, one can argue that $\mathsf{Hash}(\mathsf{pk}_i^r) \oplus \delta_{i+1}$ is uncorrelated to δ_{i+1} unless the attacker manages to compute pk_i^r and thus break the CDH assumption. Without the ROM, however, this does not follow from any standard property.

Additionally, we remark that security aside, simply avoiding the ROM wouldn't suffice to yield a suitable update homomorphism. Crucially, our generic construction relies on a homomorphic encryption scheme for which the message space roughly corresponds to the *secret-key* space. For most (efficient) additively or multiplicatively homomorphic PKE schemes, however, the message space coincides with the public-key space instead. As such, even employing different cryptographic assumptions such as pairings or isogenies do not naturally lend themselves to $\mathsf{H\text{-}UPKE}$ schemes.

The standard model schemes by Dodis *et al.* [24] circumvent this restriction by having secret keys as well as update messages that are vectors (in the length of the security parameter) of small elements. Alternative, more exotic, assumptions such as DDH groups with subgroups for which the discrete logarithm problem is easy — as e.g., considered by Castagnos and Laguillaumie [21] to build an additive homomorphic variant of ElGamal — could thus be used to circumvent this result. We remark, however, that the scheme would still need to solve circular security (e.g., by using a BHHO-like construction like the schemes in [24]).

Forgoing H-UPKE. The use of the $\mathsf{H\text{-}UPKE}$ notion appears to be a very natural instantiation of the "cumulative update" idea, but alternative constructions might exist. For instance, even without storing *secret* update information on the bulletin board, being able to sample fresh randomness and store *public* information might still help compared to the more rigid FS-PKE notion. For instance, it appears as if one could adapt the Boneh-Boyen-Goh HIBE [11] scheme to have public keys of constant size (in the number of supported epochs) by having the sender sample the public key on the fly and preparing clever update information for the receiver. Indeed, this seems to result in a (FF-)UPKE scheme where the receiver can follow on single (non-sequential) update link in the graph. However, an update graph with constant diameter and sub-linear out-degree is impossible, rendering this (naive) scheme void. Unfortunately, there appears to also be some inherent barrier in boosting such a weak FF-UPKE scheme into one that even supports following two consecutive links — without the use of some homomorphism in the encryption scheme.

6 An $(2, \mathcal{O}(\log n), \mathcal{O}(\log n))$-Update Graph

As sketched in Section 1.2, our construction is inspired by the simple family of spanner graphs that recursively join two consecutive graphs of an equal number of nodes with an overarching edge spanning from the first to the last node, with one additional node inserted at the end of each recursive step to avoid a growing indegree. We call this graph $\tilde{\mathcal{G}}_n$, where n denotes its number of nodes. See Fig. 1 (on page 8) for the example of $\tilde{\mathcal{G}}_{16}$.

Formally, the overall graph $\tilde{\mathcal{G}}$ is defined as follows.

Definition 13. *Consider the following graph* $\tilde{\mathcal{G}} := (\mathbb{N}, E)$ *with* $E := \bigcup_{k\in\mathbb{N}\cup\{0\}} E_{2^k}$*, where* $E_1 := \varnothing$ *and for* $k \in \mathbb{N}$

$$E_{2^k} := E_{2^{k-1}} \cup \left\{(i + 2^{k-1} - 1, j + 2^{k-1} - 1) \;\middle|\; (i,j) \in E_{2^{k-1}}\right\} \cup \left\{(2^k - 1, 2^k), (1, 2^k)\right\}.$$

In the remainder of the section, we prove that this construction indeed yields a $(2, \mathcal{O}(\log n), \mathcal{O}(\log n))$-update graph as needed for our FF-UPKE construction to be efficient as stated in Corollary 1. More concretely, we show the following theorem.

Theorem 5. *The graph* $\tilde{\mathcal{G}}$ *is an* $\big(2, 3\log_2(n) + 2, \log_2(n) + 1\big)$*-update graph according to Definition 11. It can be efficiently implemented according to Definition 12.*

6.1 Basic Properties of the Graph

Recall the graph $\tilde{\mathcal{G}}$ from Definition 13. In this section, we show the first couple of properties of Theorem 5. Namely, we show that $\tilde{\mathcal{G}}$ satisfies the basic properties of an update graph, has in-degree 2 and logarithmic diameter. To show the final property — namely that the active set is of logarithmic size — we then first do a detour and derive an explicit formula of $\tilde{\mathcal{G}}$.

Lemma 2. *The graph $\tilde{\mathcal{G}}$ satisfies the first two properties of an update graph, according to Definition 11.*

Proof. We first prove by induction that for all $k \in \mathbb{N} \cup \{0\}$

1. $\forall i \in [2^k - 1] : (i, i+1) \in E_{2^k}$;
2. $\forall (i,j) \in E_{2^k} : i < j$.

For $k = 0$, $[2^k - 1] = [0] = \varnothing$ and $E_{2^k} = \varnothing$, rendering both properties trivially true. Now assume $k \in \mathbb{N}$ and the induction hypothesis to hold for $k-1$. If $i \in [2^{k-1} - 1]$, then we have by the induction hypothesis that $(i, i+1) \in E_{2^{k-1}}$ and thus $(i, i+1) \in E_{2^k}$. For $2^{k-1} \leq i < 2^k - 1$ we have $(i - 2^{k-1} + 1, i - 2^{k-1} + 2) \in E_{2^{k-1}}$, since $1 \geq i - 2^{k-1} + 1 < 2^{k-1}$, and thus $(i, i+1) \in E_{2^k}$. Finally, $(2^k - 1, 2^k) \in E_{2^k}$ by definition. The second property follows directly from the induction hypothesis, as $2^k - 1 < 2^k$ and $1 < 2^k$ for $k \in \mathbb{N}$.

To conclude the proof, assume towards a contradiction that one of the properties is violated for $E = \bigcup_{k \in \mathbb{N} \cup \{0\}} E_{2^k}$. First, assume that there exists an $i \in \mathbb{N}$ such that $(i, i+1) \notin E$. This, however, contradicts that for $k \in N$ such that $2^k > i$ we have that $(i, i+1) \in E_{2^k}$ and $E_{2^k} \subset E$. Now, assume that there exists $i \geq j$ such that $(i,j) \in E$. By definition of E, there however must exists a k such that $(i,j) \in E_{2^k}$, also resulting in a contradiction to the previously proven properties. □

The Indegree. We now show that the maximal indegree of the graph $\tilde{\mathcal{G}}$ is upper bounded by two. Showing how the set of incoming edges can be efficiently computed is deferred to a later section.

Theorem 6. *The update graph $\tilde{\mathcal{G}}$ has indegree two, i.e.,*

$$\forall n \in \mathbb{N} : \deg^{\mathsf{in}}_{\tilde{\mathcal{G}}}(n) = |E^{\mathsf{in}}_{\tilde{\mathcal{G}}}(n)| \leq 2.$$

Proof. Let $k \coloneqq \lceil \log_2(n) \rceil$. We first argue that $(i,n) \in E$ (for some $i < n$) implies $(i,n) \in E_{2^k}$, i.e., that the indegree of n in $\tilde{\mathcal{G}}$ is equal to the one in $\tilde{\mathcal{G}}_{2^k}$. Assume towards a contradiction that this is not the case, i.e., there exists some $k' > k$ and $i < n$ such that $(i,n) \notin E_{2^k}$ but $(i,n) \in E_{2^{k'}}$, and consider the smallest such k', i.e., $(i,n) \notin E_{2^{k'-1}}$. Moreover, one can see that $(i,n) \notin \{(u + 2^{k'-1} - 1, v + 2^{k'-1} - 1) \mid (u,v) \in E_{2^{k'-1}}\}$ for $u + 2^{k'-1} - 1 \geq u + 2^k - 1 \geq 2^k \geq n$. Finally, as $2^{k'} > n$ we have $(i,n) \notin \{(2^{k'} - 1, 2^{k'}), (1, 2^{k'})\}$, and thus by the definition of $E_{2^{k'}}$ we obtain the contradicting $(i,n) \notin E_{2^{k'}}$.

Next, observe that $E_1 = \varnothing$ and thus every node in $\tilde{\mathcal{G}}_1$ has indegree 0. Now consider $\tilde{\mathcal{G}}_{2^k}$ for an arbitrary $k \in \mathbb{N}$ and assume as induction hypothesis that each node $n \in \mathbb{N} \setminus \{1\}$ in $\tilde{\mathcal{G}}_{2^{k-1}}$ has indegree at most 2 and node 1 has indegree 0. First, we remark that $E_{2^{k-1}} \subseteq [2^{k-1}]^2$ and $\{(u + 2^{k-1} - 1, v + 2^{k-1} - 1) \mid (u,v) \in E_{2^{k-1}}\} \subseteq \{2^{k-1}, \ldots, 2^k - 1\}^2$. Thus, node 2^k has indegree 2, from the two additional links, and the nodes in $\{1, \ldots, 2^{k-1} - 1, 2^{k-1} + 1, \ldots, 2^k - 1\}$ inherit their indegree from the respective copy of $E_{2^{k-1}}$. Since the node 2^{k-1} has moreover indegree 0 in the second copy, it follows that all nodes have indegree at most 2 and node 1 has indegree 0. □

The Diameter. Next, we prove that each prefix $\tilde{\mathcal{G}}_n$ of the graph $\tilde{\mathcal{G}}$ has a diameter $\mathcal{O}(\log n)$ diameter.

Lemma 3. *Let $k \in \mathbb{N} \setminus \{1\}$. For every $x, y \in [2^k]$ such that $x < y$ there exists a path $x \rightsquigarrow y$ in $\tilde{\mathcal{G}}_{2^k}$ such that $|x \rightsquigarrow y| \leq 3k - 5$.*

Proof. We prove the lemma using two additional claims about the shortest path from x to the middle node 2^{k-1} and from 2^{k-1} to y.

Claim. Let $k \in \mathbb{N}$. For every $y \in [2^k]$ there exists a path $1 \rightsquigarrow y$ in $\tilde{\mathcal{G}}_{2^k}$ such that $|1 \rightsquigarrow y| \leq k$.

Proof: For $k = 1$, there exists an edge $(1, 2)$, trivially satisfying the property. Now consider $k \geq 2$ and assume the property to hold for $k - 1$. If $y \leq 2^{k-1}$, then by the induction hypothesis $|1 \rightsquigarrow y| \leq k - 1$. If $y = 2^{k-1}$, then we know from the construction of $\tilde{\mathcal{G}}_{2^k}$ that there is an edge $(1, 2^k)$, i.e., $|1 \rightsquigarrow 2^k| = 1$. Finally, for $2^{k-1} < y < 2^k$, we know that there is an edge $(1, 2^{k-1})$. Moreover, by the construction of $\tilde{\mathcal{G}}_{2^k}$, we know that each path $2^{k-1} \rightsquigarrow y$ corresponds to a path $1 \rightsquigarrow y - 2^{k-1} + 1$ in $\tilde{\mathcal{G}}_{2^{k-1}}$. Thus, we know from the induction hypothesis that there exists a path such that $|2^{k-1} \rightsquigarrow y| \leq k - 1$ and, thus, $|1 \rightsquigarrow y| \leq k$. ■

Claim. Let $k \in \mathbb{N} \setminus \{1\}$. For every $x \in [2^k]$ there exists a path $x \rightsquigarrow 2^k$ in $\tilde{\mathcal{G}}_{2^k}$ such that $|x \rightsquigarrow 2^k| \leq 2k - 2$.

Proof: Let $k = 2$ and observe that there are edges $(1, 4)$, $(2, 3)$, and $(3, 4)$. Thus for every $x \in [4]$ there exits a path $|x \rightsquigarrow 4|$ of length at most 2. Now consider an arbitrary $k > 2$. If $x \leq 2^{k-1}$, then by the induction hypothesis we know that there exists a path $x \rightsquigarrow 2^{k-1}$ of length at most $2(k-1)-2$. Moreover, $\{(2^{k-1}, 2^k - 1), (2^k - 1, 2^k)\} \subseteq E_{2^k}$ and thus there exists an overall path of length $2(k - 1) - 2 + 2 = 2k - 2$. Analogously, if $2^{k-1} < x < 2^k$, then we know that a path $x \rightsquigarrow 2^k - 1$ in $\tilde{\mathcal{G}}_{2^k}$ corresponds to a path $x - 2^{k-1} + 1 \rightsquigarrow 2^{k-1}$ in $\tilde{\mathcal{G}}_{2^{k-1}}$, for which by the induction hypothesis we know that there exists one of length at most $2(k - 1) - 2$. Combining this with the edge $(2^k - 1, 2^k)$ thus immediately yields an overall path of length at most $2(k - 1) - 1 = 2k - 3$. Finally, if $x = 2^k$, there trivially exists a path of length 0. ■

We now can combine those two results to yield our claim: for any x and y in $\tilde{\mathcal{G}}_{2^k}$ we know from the first claim that there exists a path $x \rightsquigarrow 2^{k-1}$ such that $|x \rightsquigarrow 2^{k-1}| \leq k - 1$. By the recursive construction, we moreover know that a path $2^{k-1} \rightsquigarrow y$ corresponds to a path $1 \rightsquigarrow y - 2^{k-1}$ in $\tilde{\mathcal{G}}_{2^{k-1}}$, for which by the second claim there exists one of length at most $2(k - 1) - 2$, resulting in a total path of length $k - 1 + 2(k - 1) - 2 = 3k - 5$. □

Using this lemma about the diameter of the graphs $\tilde{\mathcal{G}}_{2^k}$ we now immediately yield the following theorem about the general diameter.

Theorem 7. *For every $n \in \mathbb{N}$, we have* $\operatorname{diam}(\tilde{\mathcal{G}}_n) \leq 3 \log_2(n) + 2$.

Proof. First consider the cases of $n < 4$. Clearly $\operatorname{diam}(\tilde{\mathcal{G}}_1) = 0$, $\operatorname{diam}(\tilde{\mathcal{G}}_2) = 1$, and $\operatorname{diam}(\tilde{\mathcal{G}}_3) = 2$. For $n \geq 4$, let $k \coloneqq \lceil \log_2(n) \rceil \geq 2$. By Lemma 3, we thus have

$$\operatorname{diam}(\tilde{\mathcal{G}}_n) \leq \operatorname{diam}(\tilde{\mathcal{G}}_{2^k}) \leq 3k - 5 \leq 3\lceil \log_2(n) \rceil - 2 < 3 \log_2(n) + 2,$$

concluding the proof. □

6.2 An Explicit Formula.

To provide an explicit formula of active(n) and $E^{\mathsf{in}}(n)$ (and respective algorithm to efficiently compute them), we first need to analyze the graph in some more detail. First, we observe that all non-trivial (i.e, not $(i, i + 1)$) edges have the length of a power of two minus one.

Lemma 4. *For all $(i, j) \in E$, where $\tilde{\mathcal{G}} = (\mathbb{N}, E)$, either $j = i+1$ or $j-i = 2^\ell - 1$ for some $\ell \in \{2, \ldots, \lceil \log_2(j) \rceil\}$.*

Proof. The base case $E_1 = \varnothing$ is trivial. Assume for an arbitrary $k \in \mathbb{N}$ that the property holds for $E_{2^{k-1}}$. Clearly, adding $2^{k-1} - 1$ to both the head and tail of a link $(i, j) \in E_{2^{k-1}}$ preserves its length. Moreover, $(i, j) = (2^k - 1, 2^k)$ satisfies $j = i + 1$ and $(i, j) = (1, 2^k)$ satisfies $j - i = 2^\ell - 1$ for $\ell = k$ if $k \geq 2$ and $j = i + 1$ if $k = 1$. □

Second, we provide an explicit formula for all starting points of non-trivial edges in the following lemma.

Lemma 5. *For $k \in \mathbb{N} \setminus \{1\}$ and $2 \leq \ell \leq k$, let* $\operatorname{tails}(k, \ell)$ *denote the set of vertices of $\tilde{\mathcal{G}}_{2^k}$ which have an outgoing edge of length $2^\ell - 1$, i.e.,*

$$\operatorname{tails}(k, \ell) \coloneqq \left\{ i \in [2^k] \;\middle|\; (i, i + 2^\ell - 1) \in E_{2^k} \right\}.$$

Then, we have that

$$\operatorname{tails}(k, \ell) = \left\{ i \cdot 2^\ell - \mathrm{w_H}(i) + 1 \;\middle|\; i \in \{0, \ldots, 2^{k-\ell} - 1\} \right\},$$

where $\mathrm{w_H}(i)$ denotes the Hamming weight of the binary representation of i.

Proof. We prove this using induction over k. First, for the base case consider $k = 2$, i.e., $\tilde{\mathcal{G}}_4$ for which $(1,4) \in E_4$ is obviously the only edge of length 3 and thus $\text{tails}(2,2) = \{1\}$. However, we also have $\{i \cdot 2^2 - \text{w}_\text{H}(i) + 1 \mid i \in \{0,\dots,2^{2-2}-1\}\} = \{0 \cdot 4 - w_H(0) + 1\} = \{1\}$, concluding the case $k = 2$.

Now consider an arbitrary $k > 2$. First, for the case $\ell = k$, analously to above we have that $\text{tails}(k,\ell) = \{1\} = \{i \cdot 2^\ell - \text{w}_\text{H}(i) + 1 \bigm| i \in \{0,\dots,2^0-1\}\}$. Second, consider the case $2 \le \ell < k$ and recall that

$$E_{2^k} \coloneqq E_{2^{k-1}} \cup \big\{(i + 2^{k-1} - 1, j + 2^{k-1} - 1) \bigm| (i,j) \in E_{2^{k-1}}\big\} \cup \big\{(2^k - 1, 2^k), (1, 2^k)\big\}.$$

Since the edge $(2^k - 1, 2^k)$ has length $1 < 2^\ell - 1$ and the edge $(1, 2^k)$ has length $2^k - 1 > 2^\ell - 1$, we obtain

$$\text{tails}(k,\ell) = \text{tails}(k-1,\ell) \cup \{i + 2^{k-1} - 1 \mid (i,j) \in \text{tails}(k-1,\ell)\}$$

and, thus, by the induction hypothesis

$$\begin{aligned}
\text{tails}(k,\ell) &= \{i \cdot 2^\ell - \text{w}_\text{H}(i) + 1 \mid i \in \{0,\dots,2^{k-1-\ell}-1\}\} \\
&\qquad \cup \{i \cdot 2^\ell - \text{w}_\text{H}(i) + 2^{k-1} \mid i \in \{0,\dots,2^{k-1-\ell}-1\}\} \\
&= \{i \cdot 2^\ell - \text{w}_\text{H}(i) + 1 \mid i \in \{0,\dots,2^{k-1-\ell}-1\}\} \\
&\qquad \cup \{\underbrace{(j - 2^{k-1-\ell}) \cdot 2^\ell}_{j \cdot 2^\ell - 2^{k-1}} - \text{w}_\text{H}(j - 2^{k-1-\ell}) + 2^{k-1} \mid \\
&\qquad\qquad\qquad\qquad j \in \{2^{k-1-\ell},\dots,2^{k-\ell}-1\}\} \\
&= \{i \cdot 2^\ell - \text{w}_\text{H}(i) + 1 \mid i \in \{0,\dots,2^{k-1-\ell}-1\}\} \\
&\qquad \cup \{j \cdot 2^\ell - \text{w}_\text{H}(j - 2^{k-1-\ell}) \mid j \in \{2^{k-1-\ell},\dots,2^{k-\ell}-1\}\},
\end{aligned}$$

where in the first step we substituted $i \mapsto j - 2^{k-1-\ell}$. Since $j \ge 2^{k-1-\ell}$ we moreover have $\text{w}_\text{H}(j - 2^{k-1-\ell}) = \text{w}_\text{H}(j) + 1$, which concludes the proof. □

As a corollary, we immediately obtain an explicit formula for E_{2^k}. (Recall that by definition $E_1 = \varnothing$, and by inspection it is easy to see that $E_2 = \{(1,2)\}$.)

Corollary 2. *For $k \in \mathbb{N} \setminus \{1\}$ we have*

$$E_{2^k} = \{(i, i+1) \mid i \in [2^k - 1]\} \cup \bigcup_{\ell \in \{2,\dots,k\}} \big\{\big(i \cdot 2^\ell - \text{w}_\text{H}(i) + 1, (i+1) \cdot 2^\ell - \text{w}_\text{H}(i)\big) \bigm| i \in \{0,\dots,2^{k-\ell}-1\}\big\}.$$

Additionally, we can deduce from Lemma 5 that no two edges of the graph cross.

Corollary 3. *Let $k, \ell, \ell' \in \mathbb{N}$ be arbitrary. We then have:*

(a) For any $x, x' \in \text{tails}(k,\ell)$ such that $x \ne x'$ we have $|x - x'| \ge 2^\ell - 1$. Moreover, for i such that $x = i2^\ell - \text{w}_\text{H}(i) + 1$ and $x' = i'2^\ell - \text{w}_\text{H}(i') + 1$, existing according to Lemma 5 respectively, we have $i > i'$ iff $x > x'$.
(b) For any $x \in \text{tails}(k,\ell)$ and $x' \in \text{tails}(k,\ell')$ with $x < x' < x + 2^\ell - 1$, we have $x' + 2^{\ell'} - 1 < x + 2^\ell - 1$.

Proof. We first prove Corollary 3a. First, let $i = i' + 1$. Then we have $i2^\ell - \text{w}_\text{H}(i) - (i'2^\ell - \text{w}_\text{H}(i')) = 2^\ell + \text{w}_\text{H}(i') - \text{w}_\text{H}(i' + 1) \ge 2^\ell - 1 > 0$. Hence, by an inductive argument we get $x > x' \Rightarrow i > i'$. To show the other direction, assume $x > x'$ but $i' > i$ (obviously $i' \ne i$). However as we just shown, $i' > i$ implies $x' > x$, resulting in a contradiction. Next, assume w.l.o.g. that $x > x'$. We then have

$$\begin{aligned}
x - x' &= i \cdot 2^\ell - \text{w}_\text{H}(i) + 1 - (i' \cdot 2^\ell - \text{w}_\text{H}(i') + 1) \\
&= (i - i') \cdot 2^\ell - (\text{w}_\text{H}(i) - \text{w}_\text{H}(i')) \\
&\ge (i - i') \cdot 2^\ell - (i - i') \ge 2^\ell - 1,
\end{aligned}$$

where in the third step we first used that $\text{w}_\text{H}(i) - \text{w}_\text{H}(i') \le i - i'$, as $i > i'$, concluding the first part of the proof.

Second, we show Item b. If $\ell' = \ell$, then this directly follows from Corollary 3a. If $\ell' > \ell$, then as $\text{w}_\text{H}(i \cdot 2^{\ell'-\ell}) = \text{w}_\text{H}(i)$, by Lemma 5, we have $x' \in \text{tails}(k,\ell)$ as well. Hence, by Corollary 3a $x' > x$ implies $x' \ge x + 2^\ell - 1$ and

hence the implication is trivially satisfied. Finally, consider the case of $\ell' < \ell$. Using $j \coloneqq i \cdot 2^{\ell-\ell'}$, and thus $\mathrm{w_H}(j) = \mathrm{w_H}(i)$, we can rewrite x and $x + 2^\ell - 1$ as

$$x = i \cdot 2^\ell - \mathrm{w_H}(i) - 1 = j \cdot 2^{\ell'} - \mathrm{w_H}(j) - 1, \tag{1}$$

$$x + 2^\ell - 1 < (i+1) \cdot 2^\ell - \mathrm{w_H}(i) + 1 = j \cdot 2^{\ell'} + 2^\ell - \mathrm{w_H}(j) + 1. \tag{2}$$

Hence, in order for $x < x' < x + 2^\ell - 1$ to hold, we can see i' must be in the following range:

$$i \cdot 2^{\ell-\ell'} = j < i' \leq j + 2^{\ell-\ell'} - 1 = (i+1)2^{\ell-\ell'} - 1.$$

Using Corollary 3a we know that x' is maximized if we set $i' = (i+1) \cdot 2^{\ell-\ell'} - 1$. Thus, we can derive

$$\begin{aligned} x' + 2^{\ell'} - 1 &= (i'+1) \cdot 2^{\ell'} - \mathrm{w_H}(i') \\ &= (i+1) \cdot 2^\ell - \mathrm{w_H}((i+1)2^{\ell-\ell'} - 1) \\ &= (i+1) \cdot 2^\ell - \big(\mathrm{w_H}(i \cdot 2^{\ell-\ell'}) + \mathrm{w_H}(2^{\ell-\ell'} - 1)\big) \\ &< (i+1) \cdot 2^\ell - \mathrm{w_H}(i) = x + 2^\ell - 1, \end{aligned}$$

concluding the proof. □

Active Nodes and Incoming Edges. Using Lemmas 4 and 5 and Corollary 3 explicit formulas for $\mathrm{active}(n)$ and $E^{\mathsf{in}}(n)$ can be obtained.

Lemma 6. *For the graph $\tilde{\mathcal{G}}$ and the corresponding function* tails *as defined in Lemma 5, let*

$$\mathrm{lastTail}(n,\ell) \coloneqq \max\big\{i \in \mathrm{tails}\big(\lceil \log_2(n) \rceil, \ell\big) \;\big|\; i < n\big\}$$

$$\mathrm{active}(n,\ell) \coloneqq \begin{cases} \varnothing & \text{if } \mathrm{lastTail}(n,\ell) \leq n - 2^\ell + 1, \\ \{\mathrm{lastTail}(n,\ell)\} & \text{else.} \end{cases}$$

Then, for $n \in \mathbb{N} \setminus \{1\}$ we have

$$\mathrm{active}(n) = \{1\} \cup \bigcup_{\ell \in \{2,\dots,\lceil \log_2(n) \rceil\}} \mathrm{active}(n,\ell).$$

Proof. In the following, let $k \coloneqq \lceil \log_2(n) \rceil$. We first show that $\mathrm{active}(n) \supseteq \{1\} \cup \bigcup_{\ell \in \{2,\dots,k\}} \mathrm{active}(n,\ell)$. As a first step, we recall that $(1, 2^{k+1}) \in E_{2^{k+1}}$ with $1 < n < 2^{k+1}$ and, thus, $1 \in \mathrm{active}(n)$. Next, for $\ell \in \{2,\dots,k\}$ consider $i \in \mathrm{active}(n,\ell)$. By definition this means $i = \mathrm{lastTail}(n,\ell)$ and $i > n - 2^\ell + 1$. For $j \coloneqq i + 2^\ell - 1 > n$, we thus obtain $(i,j) \in E_{2^k}$, as $i \in \mathrm{tails}(k,\ell)$, which shows that $i \in \mathrm{active}(n)$.

Second, we show that $\mathrm{active}(n) \subseteq \{1\} \cup \bigcup_{\ell \in \{2,\dots,k\}} \mathrm{active}(n,\ell)$. To this end, consider an arbitrary $i \in \mathrm{active}(n)$.

Claim. If $i \in \mathrm{active}(n)$ and $i \neq 1$, then there exists $\ell \in \{2,\dots,k\}$ such that $i = \mathrm{lastTail}(n,\ell)$ and $i > n - 2^\ell + 1$.

Proof: Recall that $i \in \mathrm{active}(n)$ means that there exists an j such that $i < n < j$ and $(i,j) \in E$. By Lemma 4 we have $j - i = 2^\ell - 1$ for some $\ell \geq 2$. Since $j > n$, this yields $i + 2^\ell - 1 > n$, which in turn is equivalent to $i > n - 2^\ell + 1$. Moreover, observe that $i \in \mathrm{tails}(\lceil \log_2(j) \rceil, \ell)$ by definition of tails. Assume towards a contradiction that $j > 2^k$. Let $k' \in \mathbb{N}$ be the smallest such that $(i,j) \in E_{2^{k'}}$. Hence, $(i,j) \notin E_{2^{k'-1}}$. Furthermore, we have $k' > k$ as $E_{2^k} \subseteq [2^k]^2$ and, thus, $i < 2^k \leq 2^{k'-1}$. As a result, we can deduce that $(i,j) \notin \{(u + 2^{k'-1} - 1, v + 2^{k'-1} - 1) \mid (u,v) \in E_{2^{k'-1}}\}$. Since moreover $j - i \geq 2$ and $i \neq 1$ we can also conclude that $(i,j) \notin \{(2^{k'} - 1, 2^{k'}), (1, 2^{k'})\}$ to arrive at the contradiction to $(i,j) \in E_{2^{k'}}$.

Hence, we have $j \leq 2^k$ and, thus, $\ell \leq k$. As $j > n$, it must however hold that $\lceil \log_2(j) \rceil = k = \lceil \log_2(n) \rceil$. Thus, we have $i \in \mathrm{tails}(k,\ell)$ and $i < n$ moreover makes it a candidate for $\mathrm{lastTail}(n,\ell)$. Using Corollary 3a we moreover know that for any $i' \in \mathrm{tails}(k,\ell)$ with $i' > i$, we have $i' \geq i + 2^\ell - 1 = j > n$. Hence, $i = \mathrm{lastTail}(n,\ell)$. ■

Using this claim, we immediately obtain that for all $i \in \mathrm{active}(n) \setminus \{1\}$, there exists $\ell \in \{2,\dots,k\}$ such that $\{i\} = \mathrm{active}(n,\ell)$, concluding the proof. □

Lemma 7. *For the graph $\tilde{\mathcal{G}}$ and the corresponding functions* tails *and* lastTail *as defined in Lemmas 5 and 6, respectively, let*

$$E^{\mathsf{in}}(n,\ell) \coloneqq \begin{cases} \{\text{lastTail}(n,\ell)\} & \text{if lastTail}(n,\ell) = n - 2^\ell + 1, \\ \varnothing & \text{else.} \end{cases}$$

Then, for $n \in \mathbb{N} \setminus \{1\}$ we have

$$E^{\mathsf{in}}(n) = \{n-1\} \cup \bigcup_{\ell \in \{2,\dots,\lceil \log_2(n) \rceil\}} E^{\mathsf{in}}(n,\ell).$$

Proof. Let $i \in E^{\mathsf{in}}(n)$, i.e., $(i,n) \in E$ and assume $n > i+1$. (If $n = i+1$, then (i,n) is obviously contained in the right hand set.) By Lemma 4 there exists $\ell \geq 2$ such that $n - i = 2^\ell - 1$. For $k \coloneqq \lceil \log_2(n) \rceil$, we thus have $i \in \text{tails}(k,\ell)$. Using Corollary 3a, we moreover know that for $i' > i$, such that $i' \in \text{tails}(k,\ell)$, it holds that $i' \geq i + 2^\ell - 1 = n$. As a consequence, we have $i = \text{lastTail}(n,\ell)$, and since $i = n - 2^\ell + 1$, thus $E^{\mathsf{in}}(n,\ell) = \{i\}$.

For the other direction, first observe that $\{n-1\} \in E^{\mathsf{in}}(n)$ follows directly by the update graph property of $\tilde{\mathcal{G}}$. Now consider an arbitrary element of the union, i.e., $i \in E^{\mathsf{in}}(n,\ell)$ for some $\ell > 2$. By definition we thus have $i = \text{lastTail}(n,\ell) \in \text{tails}(\lceil \log_2(n) \rceil, \ell)$ and $i = n - 2^\ell + 1$. Hence, by the definition of tails, $(i,n) \in E_{2^{\lceil \log_2(n) \rceil}} \subset E$, which is equivalent to $i \in E^{\mathsf{in}}(n)$. □

6.3 Implementing the Graph.

It remains to implement our update graph $\tilde{\mathcal{G}}$. Recall that we require an implementation of an (α,β,γ)-update graph to provide the following two algorithms:

- $(\mathsf{In},\mathsf{A}) \leftarrow \tilde{\mathcal{G}}.\mathsf{Eval}(i)$ which computes $\mathsf{In} = E^{\mathsf{in}}(n)$ and $\mathsf{A} = \text{active}(n)$ in $\mathcal{O}(\mathsf{poly}(\log n))$ time,
- $\vec{p} \leftarrow \tilde{\mathcal{G}}.\mathsf{Path}(i,j)$ which computes a path from node i to j such that $|\vec{p}| \leq \text{diam}(j) \leq \beta(j)$ in $\mathcal{O}(\mathsf{poly}(\beta(j)j))$.

Our implementation makes use of the helper $\text{lastTail}(n,\ell)$, which returns the largest index $x < n$ that has an outgoing edge of length $2^\ell - 1$. (Note that according to the construction, all edges are of length $2^\ell - 1$ for some $\ell \in \mathbb{N}$.) The concrete implementation of $\text{lastTail}(n,\ell)$ is based on an explicit formula of the graph outlined in Section 6.2.

The $\tilde{\mathcal{G}}.\mathsf{Path}(i,j)$ algorithm depicted in Fig. 9 works by greedily always taking the longest outgoing edge that does not go past node j. To this end, note that if there is an outgoing edge of length $2^\ell - 1$ from node i, then $\text{lastTail}(i + 2^\ell - 1, \ell) = i$. This is due to the fact that in $\tilde{\mathcal{G}}$ edges of equal length never cross. Thus, any node k with $i < k < i + 2^\ell - 1$ does not have an outgoing edge of length $2^\ell - 1$ and, hence, i is the largest index that satisfies the condition of lastTail.

The $\tilde{\mathcal{G}}.\mathsf{Eval}(n)$ needs to output the set of incoming edges $E^{\mathsf{in}}_{\tilde{\mathcal{G}}}(n)$ to node n as well as the set of active nodes $\text{active}_{\tilde{\mathcal{G}}}(n)$. To this end, it considers all possible edge lengths $2^\ell - 1$ with $\ell \leq \lceil \log_2(n) \rceil$. (Recall that any active edge not present in the subgraph $\tilde{\mathcal{G}}_{\lceil \log_2(n) \rceil}$ of the first $\lceil \log_2(n) \rceil$ nodes will start at node 1 that can be handled as a special case.) If $\text{lastTail}(n,\ell)$ equals $n - 2^\ell + 1$ then by the same argument as in $\tilde{\mathcal{G}}.\mathsf{Path}(i,j)$ there is an incoming edge from $n - 2^\ell + 1$. If $\text{lastTail}(n,\ell) > n - 2^\ell + 1$, then an edge of length $2^\ell - 1$ starts at $\text{lastTail}(n,\ell)$ and crosses node n, thus belonging to the active set. As no edges of the same length cross, those are moreover all active nodes.

Theorem 8. *The algorithms in Fig. 9 implement a $\big(2, 3\log_2(n) + 2, \log_2(n) + 1\big)$-update graph.*

Proof. We divide the proof into two parts about the respective algorithms.

Claim. The algorithm $\tilde{\mathcal{G}}.\mathsf{Eval}(n)$ outputs the sets $E^{\mathsf{in}}_{\tilde{\mathcal{G}}}(n)$ and $\text{active}_{\tilde{\mathcal{G}}}(n)$ in $\mathcal{O}(\mathsf{poly}(\log n))$ time.

Proof: It is easy to see that $\tilde{\mathcal{G}}.\mathsf{Eval}(n)$ makes $\mathcal{O}(\log n)$ invocations to the helper algorithm $\text{lastTail}(n,\ell)$, which in turn has running time $\mathcal{O}(\mathsf{poly}(\log n))$. For correctness, we moreover immediately get by Lemmas 6 and 7 that the output is correct as long as lastTail implements the function defined in Lemma 6. To this end, consider $\text{lastTail}(n,\ell)$ as defined in Lemma 6. Using Lemma 5 we can conclude that for $x \coloneqq \text{lastTail}(n,\ell)$ we have

$$x = \text{lastTail}(n,\ell) = \max\{i2^\ell - \text{w}_\text{H}(i) + 1 \mid i < 2^{\log_2(n) - \ell} \wedge i2^\ell - \text{w}_\text{H}(i) + 1 < n\}.$$

Protocol Update Graph

```
Ğ.Eval(n)
  if n = 1 then
      return (∅, ∅)
  k ← ⌈log_2(n)⌉
  A ← {1}
  In ← {n − 1}
  for ℓ ← 2, . . . , k do
      x ← lastTail(n, ℓ)
      if x > n − 2^ℓ + 1 then
          A ← A ∪ {x}
      else if x = n − 2^ℓ + 1 then
          In ← In ∪ {x}
  return (In, A)
```

```
Ğ.Path(i, j)
  e⃗ ← ∅
  while i < j do
      // greedily take longest link
      for ℓ ∈ k, k − 1, . . . , 2 do
          p ← i + 2^ℓ − 1
          if lastTail(p, ℓ) = i then
              e⃗ ← e⃗ ‖ (i, p)
              i ← p
              continue // jump to header of while
      // no non-trivial link found
      e⃗ ← e⃗ ‖ (i, i + 1)
      i ← i + 1
  return e⃗
```

```
helper lastTail(n, ℓ)
  k ← ⌈log_2(n)⌉
  x_max ← 0
  for q ← 0, . . . , k + 1 do
      i ← ⌊(n − 1 + q)/2^ℓ⌋
      x ← i2^ℓ − w_H(i) + 1
      if x ≥ x_max ∧ x < n ∧ 2^{k−ℓ} then
          x_max ← x
  return x_max
```

Fig. 9: An implementation of the $(2, \mathcal{O}(\log n), \mathcal{O}(\log n))$-update graph $\tilde{\mathcal{G}}$.

First, for $i_{min} = \left\lfloor \frac{n-2}{2^\ell} \right\rfloor$, we have

$$i_{min} 2^\ell - \mathrm{w_H}(i) + 1 \leq n - 2 - \mathrm{w_H}(i_{min}) + 1 < n,$$

and

$$\left\lfloor \frac{n-2}{2^\ell} \right\rfloor \leq \frac{n-2}{2^\ell} \leq \frac{2^k - 2}{2^\ell} < 2^{k-\ell},$$

making i_{min} a valid candidate for the maximum.. Second, we can observe that

$$i = \frac{x + \mathrm{w_H}(i) - 1}{2^\ell} \leq \frac{x + \mathrm{w_H}(n) - 1}{2^\ell} \leq \frac{x + k - 1}{2^\ell},$$

Hence, if i^* denotes the one maximizing $\mathrm{lastTail}(n, \ell)$, we have

$$\left\lfloor \frac{n-2}{2^\ell} \right\rfloor \leq i^* \leq \left\lfloor \frac{n+k-1}{2^\ell} \right\rfloor,$$

showing that our implementation of $\mathrm{lastTail}(n, \ell)$ is correct. ■

Claim. The greedy algorithm implemented by $\tilde{\mathcal{G}}.\mathsf{Path}(i, j)$ returns the shortest path from i to j.

Proof: This directly follows from Corollary 3b, stating that no two edges in the graph cross. In other words, if we are currently at node u and (u, v) is the longest outgoing edge, then it is never beneficial to not take this edge, given that there cannot be an edge starting at $u < w < v$ that skips v. ■

6.4 Proof of Theorem 5

Theorem 5. *The graph $\tilde{\mathcal{G}}$ is an $\big(2, 3\log_2(n) + 2, \log_2(n) + 1\big)$-update graph according to Definition 11. It can be efficiently implemented according to Definition 12.*

Proof. The first two properties of Definition 11 have been proven as a part of Lemma 2. The third and fourth properties have been proven as Theorems 6 and 7, respectively. The final property directly follows from Lemma 6 as $|\mathrm{active}(n, \ell)| \leq 1$, by definition, and $|\{2, \ldots, \lceil \log_2(n) \rceil\}| \leq \log_2(n)$. The graph being efficiently implementable has been shown in Theorem 8. □

7 Conclusions and Open Problems

We identified *fast-forwarding* as a compelling property of forward-secure encryption, and have shown that in the practically relevant *bulletin-board model* fast-forwarding can be obtained at little additional cost. First, we have constructed a fast-forwardable stream cipher that maintains a constant local state and has a constant running

time per update operation. This essentially matches the efficiency of non-fast-forwardable stream ciphers at the cost of constant communication complexity with the bulletin board per update.

Second, we presented a generic construction of a fast-forwardable updatable public-key encryption scheme from a novel primitive of an update-homomorphic UPKE scheme. This bridges the gap between forward-secure PKE, for which fast-forwardability is the norm, and its more efficient cousin UPKE, where none of the existing schemes were fast-forwardable. As a feasibility result, we presented instantiations based on the DDH and LWE assumptions, respectively.

While neither instantiation is truly practical, we believe that our novel construction of FF-UPKE could ultimately lead to constructions significantly outperforming those of forward-secure PKE, resolving the dilemma of practical public-key encryption to having to choose between forward-secrecy and fast-forwarding. Accordingly, this leaves the construction of efficient update-homomorphic UPKE schemes as an intriguing problem, demonstrating that while highly practical UPKE schemes are known to exist in the ROM, the search for efficient schemes in the standard model may be of interest for the sake of exhibiting homomorphic properties typically unknown to ROM constructions.

Acknowledgments

We would like to thank Michael Elkin for a useful discussion about update graphs and bringing [44] to our attention.

References

1. Abdalla, M., Reyzin, L.: A new forward-secure digital signature scheme. In: Okamoto, T. (ed.) ASIACRYPT 2000. LNCS, vol. 1976, pp. 116–129. Springer, Heidelberg (Dec 2000). https://doi.org/10.1007/3-540-44448-3_10
2. Agrawal, S., Boneh, D., Boyen, X.: Efficient lattice (H)IBE in the standard model. In: Gilbert, H. (ed.) EUROCRYPT 2010. LNCS, vol. 6110, pp. 553–572. Springer, Heidelberg (May / Jun 2010). https://doi.org/10.1007/978-3-642-13190-5_28
3. Alwen, J., Coretti, S., Dodis, Y., Tselekounis, Y.: Security analysis and improvements for the IETF MLS standard for group messaging. In: Micciancio, D., Ristenpart, T. (eds.) CRYPTO 2020, Part I. LNCS, vol. 12170, pp. 248–277. Springer, Heidelberg (Aug 2020). https://doi.org/10.1007/978-3-030-56784-2_9
4. Alwen, J.: Subject: [MLS] UPKE for X25519/X448. MLS Mailing List (21 October 2019 21:20UTC), https://mailarchive.ietf.org/arch/msg/mls/y5ikXqQh7VAojXrNSt9odxgNVmE
5. Anderson, R.: Invited lecture. Fourth Annual Conference on Computer and Communications Security, ACM (1997)
6. Aviram, N., Gellert, K., Jager, T.: Session resumption protocols and efficient forward security for TLS 1.3 0-RTT. Journal of Cryptology **34**(3), 20 (Jul 2021). https://doi.org/10.1007/s00145-021-09385-0
7. Bellare, M., Miner, S.K.: A forward-secure digital signature scheme. In: Wiener, M.J. (ed.) CRYPTO'99. LNCS, vol. 1666, pp. 431–448. Springer, Heidelberg (Aug 1999). https://doi.org/10.1007/3-540-48405-1_28
8. Bellare, M., Yee, B.S.: Forward-security in private-key cryptography. In: Joye, M. (ed.) CT-RSA 2003. LNCS, vol. 2612, pp. 1–18. Springer, Heidelberg (Apr 2003). https://doi.org/10.1007/3-540-36563-X_1
9. Blum, L., Blum, M., Shub, M.: Comparison of two pseudo-random number generators. In: Chaum, D., Rivest, R.L., Sherman, A.T. (eds.) CRYPTO'82. pp. 61–78. Plenum Press, New York, USA (1982)
10. Boneh, D., Boyen, X.: Efficient selective-ID secure identity based encryption without random oracles. In: Cachin, C., Camenisch, J. (eds.) EUROCRYPT 2004. LNCS, vol. 3027, pp. 223–238. Springer, Heidelberg (May 2004). https://doi.org/10.1007/978-3-540-24676-3_14
11. Boneh, D., Boyen, X., Goh, E.J.: Hierarchical identity based encryption with constant size ciphertext. In: Cramer, R. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 440–456. Springer, Heidelberg (May 2005). https://doi.org/10.1007/11426639_26
12. Boneh, D., Eskandarian, S., Kim, S., Shih, M.: Improving speed and security in updatable encryption schemes. In: ASIACRYPT 2020, Part III. pp. 559–589. LNCS, Springer, Heidelberg (Dec 2020). https://doi.org/10.1007/978-3-030-64840-4_19
13. Boneh, D., Lewi, K., Montgomery, H.W., Raghunathan, A.: Key homomorphic PRFs and their applications. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013, Part I. LNCS, vol. 8042, pp. 410–428. Springer, Heidelberg (Aug 2013). https://doi.org/10.1007/978-3-642-40041-4_23
14. Boneh, D., Waters, B.: Constrained pseudorandom functions and their applications. In: Sako, K., Sarkar, P. (eds.) ASIACRYPT 2013, Part II. LNCS, vol. 8270, pp. 280–300. Springer, Heidelberg (Dec 2013). https://doi.org/10.1007/978-3-642-42045-0_15
15. Boyd, C., Davies, G.T., Gjøsteen, K., Jiang, Y.: Fast and secure updatable encryption. In: Micciancio, D., Ristenpart, T. (eds.) CRYPTO 2020, Part I. LNCS, vol. 12170, pp. 464–493. Springer, Heidelberg (Aug 2020). https://doi.org/10.1007/978-3-030-56784-2_16

16. Brakerski, Z., Lombardi, A., Segev, G., Vaikuntanathan, V.: Anonymous IBE, leakage resilience and circular security from new assumptions. In: Nielsen, J.B., Rijmen, V. (eds.) EUROCRYPT 2018, Part I. LNCS, vol. 10820, pp. 535–564. Springer, Heidelberg (Apr / May 2018). https://doi.org/10.1007/978-3-319-78381-9_20
17. Brakerski, Z., Vaikuntanathan, V.: Efficient fully homomorphic encryption from (standard) LWE. SIAM J. Comput. **43**(2), 831–871 (2014). https://doi.org/10.1137/120868669, https://doi.org/10.1137/120868669
18. Canetti, R., Halevi, S., Katz, J.: A forward-secure public-key encryption scheme. In: Biham, E. (ed.) EUROCRYPT 2003. LNCS, vol. 2656, pp. 255–271. Springer, Heidelberg (May 2003). https://doi.org/10.1007/3-540-39200-9_16
19. Canetti, R., Halevi, S., Katz, J.: Chosen-ciphertext security from identity-based encryption. In: Cachin, C., Camenisch, J. (eds.) EUROCRYPT 2004. LNCS, vol. 3027, pp. 207–222. Springer, Heidelberg (May 2004). https://doi.org/10.1007/978-3-540-24676-3_13
20. Cash, D., Hofheinz, D., Kiltz, E., Peikert, C.: Bonsai trees, or how to delegate a lattice basis. Journal of Cryptology **25**(4), 601–639 (Oct 2012). https://doi.org/10.1007/s00145-011-9105-2
21. Castagnos, G., Laguillaumie, F.: Linearly homomorphic encryption from DDH. In: Nyberg, K. (ed.) CT-RSA 2015. LNCS, vol. 9048, pp. 487–505. Springer, Heidelberg (Apr 2015). https://doi.org/10.1007/978-3-319-16715-2_26
22. Derler, D., Jager, T., Slamanig, D., Striecks, C.: Bloom filter encryption and applications to efficient forward-secret 0-RTT key exchange. In: Nielsen, J.B., Rijmen, V. (eds.) EUROCRYPT 2018, Part III. LNCS, vol. 10822, pp. 425–455. Springer, Heidelberg (Apr / May 2018). https://doi.org/10.1007/978-3-319-78372-7_14
23. Diffie, W., van Oorschot, P.C., Wiener, M.J.: Authentication and authenticated key exchanges. Designs, Codes and Cryptography **2**(2), 107–125 (Jun 1992)
24. Dodis, Y., Karthikeyan, H., Wichs, D.: Updatable public key encryption in the standard model. In: Theory of Cryptography. Springer International Publishing (2021)
25. Dodis, Y., Katz, J., Xu, S., Yung, M.: Key-insulated public key cryptosystems. In: Knudsen, L.R. (ed.) EUROCRYPT 2002. LNCS, vol. 2332, pp. 65–82. Springer, Heidelberg (Apr / May 2002). https://doi.org/10.1007/3-540-46035-7_5
26. Dodis, Y., Katz, J., Xu, S., Yung, M.: Strong key-insulated signature schemes. In: Desmedt, Y. (ed.) PKC 2003. LNCS, vol. 2567, pp. 130–144. Springer, Heidelberg (Jan 2003). https://doi.org/10.1007/3-540-36288-6_10
27. Döttling, N., Garg, S.: From selective IBE to full IBE and selective HIBE. In: Kalai, Y., Reyzin, L. (eds.) TCC 2017, Part I. LNCS, vol. 10677, pp. 372–408. Springer, Heidelberg (Nov 2017). https://doi.org/10.1007/978-3-319-70500-2_13
28. Döttling, N., Garg, S.: Identity-based encryption from the Diffie-Hellman assumption. In: Katz, J., Shacham, H. (eds.) CRYPTO 2017, Part I. LNCS, vol. 10401, pp. 537–569. Springer, Heidelberg (Aug 2017). https://doi.org/10.1007/978-3-319-63688-7_18
29. Everspaugh, A., Paterson, K.G., Ristenpart, T., Scott, S.: Key rotation for authenticated encryption. In: Katz, J., Shacham, H. (eds.) CRYPTO 2017, Part III. LNCS, vol. 10403, pp. 98–129. Springer, Heidelberg (Aug 2017). https://doi.org/10.1007/978-3-319-63697-9_4
30. Gentry, C., Peikert, C., Vaikuntanathan, V.: Trapdoors for hard lattices and new cryptographic constructions. In: Ladner, R.E., Dwork, C. (eds.) 40th ACM STOC. pp. 197–206. ACM Press (May 2008). https://doi.org/10.1145/1374376.1374407
31. Gentry, C., Silverberg, A.: Hierarchical ID-based cryptography. In: Zheng, Y. (ed.) ASIACRYPT 2002. LNCS, vol. 2501, pp. 548–566. Springer, Heidelberg (Dec 2002). https://doi.org/10.1007/3-540-36178-2_34
32. Goldreich, O., Goldwasser, S., Micali, S.: How to construct random functions. Journal of the ACM **33**(4), 792–807 (Oct 1986)
33. Green, M.D., Miers, I.: Forward secure asynchronous messaging from puncturable encryption. In: 2015 IEEE Symposium on Security and Privacy. pp. 305–320. IEEE Computer Society Press (May 2015). https://doi.org/10.1109/SP.2015.26
34. Günther, C.G.: An identity-based key-exchange protocol. In: Quisquater, J.J., Vandewalle, J. (eds.) EUROCRYPT'89. LNCS, vol. 434, pp. 29–37. Springer, Heidelberg (Apr 1990). https://doi.org/10.1007/3-540-46885-4_5
35. Horwitz, J., Lynn, B.: Toward hierarchical identity-based encryption. In: Knudsen, L.R. (ed.) EUROCRYPT 2002. LNCS, vol. 2332, pp. 466–481. Springer, Heidelberg (Apr / May 2002). https://doi.org/10.1007/3-540-46035-7_31
36. Itkis, G., Reyzin, L.: Forward-secure signatures with optimal signing and verifying. In: Kilian, J. (ed.) CRYPTO 2001. LNCS, vol. 2139, pp. 332–354. Springer, Heidelberg (Aug 2001). https://doi.org/10.1007/3-540-44647-8_20
37. Jaeger, J., Stepanovs, I.: Optimal channel security against fine-grained state compromise: The safety of messaging. In: Shacham, H., Boldyreva, A. (eds.) CRYPTO 2018, Part I. LNCS, vol. 10991, pp. 33–62. Springer, Heidelberg (Aug 2018). https://doi.org/10.1007/978-3-319-96884-1_2
38. Jost, D., Maurer, U., Mularczyk, M.: Efficient ratcheting: Almost-optimal guarantees for secure messaging. In: Ishai, Y., Rijmen, V. (eds.) EUROCRYPT 2019, Part I. LNCS, vol. 11476, pp. 159–188. Springer, Heidelberg (May 2019). https://doi.org/10.1007/978-3-030-17653-2_6
39. Klooß, M., Lehmann, A., Rupp, A.: (R)CCA secure updatable encryption with integrity protection. In: Ishai, Y., Rijmen, V. (eds.) EUROCRYPT 2019, Part I. LNCS, vol. 11476, pp. 68–99. Springer, Heidelberg (May 2019). https://doi.org/10.1007/978-3-030-17653-2_3

40. Kozlov, A., Reyzin, L.: Forward-secure signatures with fast key update. In: Cimato, S., Galdi, C., Persiano, G. (eds.) SCN 02. LNCS, vol. 2576, pp. 241–256. Springer, Heidelberg (Sep 2003). https://doi.org/10.1007/3-540-36413-7_18
41. Lehmann, A., Tackmann, B.: Updatable encryption with post-compromise security. In: Nielsen, J.B., Rijmen, V. (eds.) EUROCRYPT 2018, Part III. LNCS, vol. 10822, pp. 685–716. Springer, Heidelberg (Apr / May 2018). https://doi.org/10.1007/978-3-319-78372-7_22
42. Malkin, T., Micciancio, D., Miner, S.K.: Efficient generic forward-secure signatures with an unbounded number of time periods. In: Knudsen, L.R. (ed.) EUROCRYPT 2002. LNCS, vol. 2332, pp. 400–417. Springer, Heidelberg (Apr / May 2002). https://doi.org/10.1007/3-540-46035-7_27
43. Poettering, B., Rösler, P.: Towards bidirectional ratcheted key exchange. In: Shacham, H., Boldyreva, A. (eds.) CRYPTO 2018, Part I. LNCS, vol. 10991, pp. 3–32. Springer, Heidelberg (Aug 2018). https://doi.org/10.1007/978-3-319-96884-1_1
44. Solomon, S., Elkin, M.: Balancing degree, diameter and weight in euclidean spanners. In: de Berg, M., Meyer, U. (eds.) Algorithms - ESA 2010, 18th Annual European Symposium, Liverpool, UK, September 6-8, 2010. Proceedings, Part I. Lecture Notes in Computer Science, vol. 6346, pp. 48–59. Springer (2010). https://doi.org/10.1007/978-3-642-15775-2_5, https://doi.org/10.1007/978-3-642-15775-2_5
45. Sun, S., Yuan, X., Liu, J.K., Steinfeld, R., Sakzad, A., Vo, V., Nepal, S.: Practical backward-secure searchable encryption from symmetric puncturable encryption. In: Lie, D., Mannan, M., Backes, M., Wang, X. (eds.) ACM CCS 2018. pp. 763–780. ACM Press (Oct 2018). https://doi.org/10.1145/3243734.3243782
46. Wei, J., Chen, X., Wang, J., Hu, X., Ma, J.: Forward-secure puncturable identity-based encryption for securing cloud emails. In: Sako, K., Schneider, S., Ryan, P.Y.A. (eds.) ESORICS 2019, Part II. LNCS, vol. 11736, pp. 134–150. Springer, Heidelberg (Sep 2019). https://doi.org/10.1007/978-3-030-29962-0_7

A Simple Constructions

A.1 GGM as an FF-PRG

In this section, we outline the adaptation of the basic GGM construction to a fast-forwardable PRG (see Definition 2). To this end, we adapt the template [7,42] (for building forward-secure signature schemes) to the GGM tree.

The scheme. The idea is to use the i-th leaf value $v_{\mathsf{leaf}(i)}$ of the GGM tree as the i-th PRG output. In epoch i one then additionally stores the *right copath* of $v_{\mathsf{leaf}(i)}$ — the set of nodes u such that u is a right child, $u.\mathsf{parent}$ is an ancestor of $v_{\mathsf{leaf}(i)}$, and u is not an ancestor of $v_{\mathsf{leaf}(i)}$ (i.e., is not on its direct path). From these, at most $\log(n)$ internal nodes of the GGM tree one can derive precisely all the secrets for periods $j \geq i$ using at most $\log(n)$ PRG evaluations, giving us the desired sublinear (actually, logarithmic) fast-forward property. At the same time, however, none of the previous epoch's secrets can be derived, ensuring forward secrecy.

In Fig. 10 we present a formal definition of this FF-PRG protocol for 2^{h} outputs using a complete binary tree of depth h. The Init algorithm thus takes an initial seed and uses a length-doubling PRG to derive the leftmost leaf, storing all the derived right children as part of the local state. The Leap algorithm, to jump from epoch i to j, first finds the unique node on the right copath that is an ancestor of the j-th leaf, which it can then derive from there. To update the stored right copath, observe that the right copaths of the i-th and j-th leaves share the part above this node. Thus, the algorithm deletes all nodes below that node and stores the ones obtained when deriving the new leaf instead. Finally, the Update algorithm is just a special case of the Leap algorithm.

Note that none of the algorithms uses the bulletin board to either store or receive data. As a result, both Update-Idx and Leap-Idx return an empty set of indices to access.

Analysis. In the following we argue that the aforementioned construction is a secure FF-PRG according to Definitions 3 and 4. More concretely, we claim that it satisfies the following lemma stated in Section 1.

Lemma 1 (Informal). *The template [7,42] (for building forward-secure signature schemes) can be adapted to the Goldreich-Goldwasser-Micali (GGM) construction [32] to build a forward-secure PRG with the fast-forward property. If n denotes the maximal number of epochs, then the scheme stores $\mathcal{O}(\log(n))$ seeds as local state, and sequential updating as well as fast-forwarding from epoch i to $j > i$ take $\mathcal{O}(\log(n))$ PRG expansions.*

Protocol GGM based FF-PRG

The parameter h specifies the GGM tree's depth, i.e., the PRG supports up to 2^{h} epochs.

$\underline{\mathsf{Init}(\mathsf{key})}$
- $\mathsf{Seeds}[\cdot] \leftarrow \bot$
- $\mathsf{s}_l \leftarrow \mathsf{key}$
- **for** $i \leftarrow 1, \ldots, \mathsf{h}$ **do** // Expand leftmost path
  - $(\mathsf{s}_l, \mathsf{s}_r) \leftarrow \mathsf{PRG.Expand}(\mathsf{s}_l)$
  - // Keep co-path nodes in state
  - $\mathsf{idx}_r \leftarrow \mathsf{rightChild}(i)$
  - $\mathsf{Seeds}[\mathsf{idx}_r] \leftarrow \mathsf{s}_r$
- $\mathsf{st}_1 \leftarrow (1, \mathsf{Seeds})$
- **return** $(\mathsf{st}_1, \mathsf{R}_1, \varnothing)$

$\underline{\mathsf{Update\text{-}Idx}(i)}$
- **return** $\varnothing$

$\underline{\mathsf{Leap\text{-}Idx}(i, j)}$
- **return** $\varnothing$

$\underline{\mathsf{Update}(\mathsf{st}_i, \mathsf{BB}_{\mathsf{I}_i})}$
- **parse** $(i, \mathsf{Seeds}) \leftarrow \mathsf{st}_i$
- **return** $\mathsf{Leap}(\mathsf{st}_i, i+1, \mathsf{BB}_{\mathsf{I}_i})$

$\underline{\mathsf{Leap}(\mathsf{st}_i, j, \mathsf{BB}_{\mathsf{I}_{i,j}})}$
- **parse** $(i, \mathsf{Seeds}) \leftarrow \mathsf{st}_i$
- $\mathsf{idx} \leftarrow \mathsf{leaf}(i)$
- $\mathsf{idx}' \leftarrow \mathsf{leaf}(j)$
- **repeat**// Find copath ancestor of idx'
  - $\mathsf{idx} \leftarrow \mathsf{rCoPath}(\mathsf{idx})$
  - $\mathsf{s} \leftarrow \mathsf{Seeds}[\mathsf{idx}]$
  - $\mathsf{Seeds}[\mathsf{idx}] \leftarrow \bot$ // Delete for forward secrecy
  - $\mathsf{idx} \leftarrow \mathsf{idx}'$
- **until** $\mathsf{isAncestor}(\mathsf{idx}, \mathsf{idx}')$
- **while** $\mathsf{idx} \neq \mathsf{idx}'$ **do** // Derive path to idx'
  - **if** $\mathsf{isAncestor}(\mathsf{idx}+1, \mathsf{idx}')$ **then** // Go down left
    - $(\mathsf{s}, \mathsf{s}_r) \leftarrow \mathsf{PRG.Expand}(\mathsf{s})$
    - $\mathsf{idx}_r \leftarrow \mathsf{rightChild}(\mathsf{idx})$
    - $\mathsf{Seeds}[\mathsf{idx}_r] \leftarrow \mathsf{s}_r$ // Store right copath
    - $\mathsf{idx} \leftarrow \mathsf{idx}+1$
  - **else** // Go down right
    - $(\cdot, \mathsf{s}) \leftarrow \mathsf{PRG.Expand}(\mathsf{s})$
    - $\mathsf{idx} \leftarrow \mathsf{rightChild}(\mathsf{idx})$
- $\mathsf{st}_j \leftarrow (j, \mathsf{Seeds})$
- **return** $(\mathsf{st}_j, \mathsf{s}, \varnothing)$

Fig. 10: The GGM-based FF-PRG does not make use of the bulletin board.

Proof (Sketch). The efficiency claim follows directly by inspection of the construction depicted in Fig. 10. For correctness, we need to argue that fast-forwarding yields both the same state and output as repeatedly sequential updating. First, we argue that both methods end up outputting the same j-th leaf value for epoch j. To this end, we observe that for any $i < j$, the j-th leaf is a descendent of one of the i-th right copath's nodes. (In general any leaf $j \neq i$ is one one of the i-th copath's nodes with $j > i$ ensuring that this holds true when restricting to the *right* copath.) Second, we observe that for both methods the state ends up being the j-th right copath.

For security, we first observe that due to the scheme being deterministic we only need to consider at most one corruption. W.l.o.g. assume that the adversary corrupts the j-th state. As argued above, this state consists of the j-th right copath, from which all leaves $k > j$ can be reached but none of the leaves $i < j$. Hence, security follows directly from the security of the underlying PRG used to expand the GGM tree. □

A.2 A HIBE Based FF-UPKE

We can adapt the previous section's scheme to a fast-forwardable UPKE scheme by the use of hierarchical identity-based encryption (HIBE) instead of the PRG. In a nutshell, the scheme replaces the GGM tree with the HIBE secret-key tree maintained by the receiver. Before outlining the scheme, let us thus briefly introduce HIBE in a bit more detail.

Hierarchical Identity-Based Encryption. A hierarchical identity-based encryption (HIBE) [35,31] scheme is an encryption scheme, that by using the master public key allows encrypting to a specific identity vector $\vec{\mathrm{id}}$, such that the message can only be decrypted using a secret key for that identity $\vec{\mathrm{id}}$. Those secret keys can be hierarchically derived from the master secret key. That is, from the secret key for $\vec{\mathrm{id}}$, secret keys for $\vec{\mathrm{id}} \parallel \vec{\mathrm{id}}'$, for arbitrary suffixes $\vec{\mathrm{id}}'$, can be derived. The HIBE secret keys can be seen as a tree, where the master secret key, corresponding to the empty $\vec{\mathrm{id}}$, is the root, and keys at depth k correspond to $\vec{\mathrm{id}}$'s of length k. A secret key allows decrypting ciphertexts encrypted for keys in its subtree, and only those.

Formally, a HIBE scheme is a tuple of algorithms $\mathsf{HIBE} = (\mathsf{HIBE.setup}, \mathsf{HIBE.kg}, \mathsf{HIBE.enc}, \mathsf{HIBE.dec})$ with the following syntax.

- *Setup:* $(\mathsf{mpk}, \mathsf{hsk}_{()}) \leftarrow \mathsf{HIBE.setup}$ samples a fresh master public key and the corresponding root secret key.
- *Key Generation:* $\mathsf{hsk}_{\vec{\mathrm{id}}\|\mathrm{id}} \leftarrow \mathsf{HIBE.kg}(\mathsf{hsk}_{\vec{\mathrm{id}}}, \mathrm{id})$ outputs a lower-level secret key. For compactness, we write $\mathsf{hsk}_{\vec{\mathrm{id}}\|\vec{\mathrm{id}}'} \leftarrow \mathsf{HIBE.kg}(\mathsf{hsk}_{\vec{\mathrm{id}}}, \vec{\mathrm{id}}')$ to denote the iterative derivation of $\mathsf{hsk}_{\vec{\mathrm{id}}\|\vec{\mathrm{id}}'}$ using each $\mathrm{id} \in \vec{\mathrm{id}}'$.

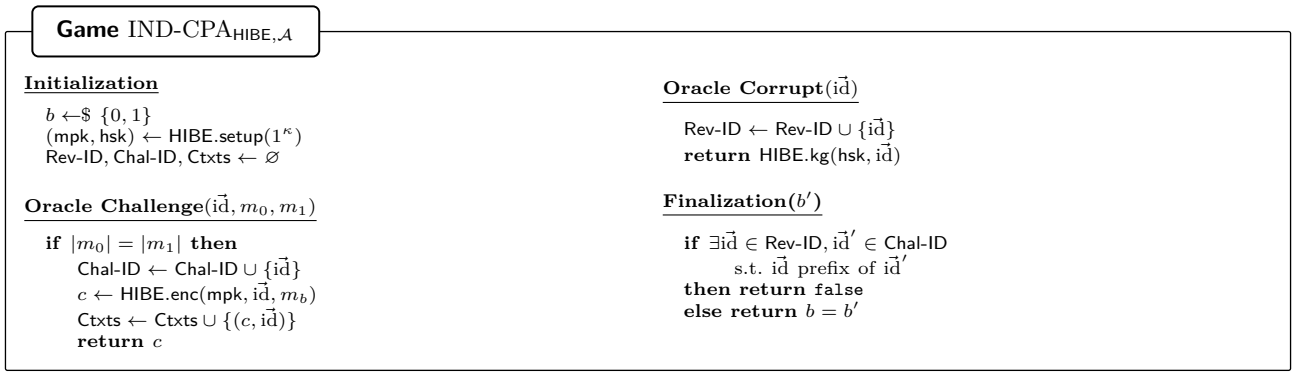

Fig. 11: The IND-CPA game for HIBE.

- *Encryption:* $c \leftarrow \mathsf{HIBE.enc}(\mathsf{mpk}, \vec{\mathrm{id}}, m)$ encrypts m for identity vector $\vec{\mathrm{id}}$.
- *Decryption:* $m \leftarrow \mathsf{HIBE.dec}(\mathsf{hsk}_{\vec{\mathrm{id}}}, c)$ decrypts c.

We require that HIBE schemes satisfy IND-CPA security, as captured by the game depicted in Fig. 11. It is the standard IND-CPA game with an additional corrupt oracle, which outputs secret keys for identity vectors chosen by the adversary. To disable "trivial wins", the game stores the set Chal-ID of identity vectors used in challenge queries and the set Rev-ID of identity vectors used in corrupt queries, and checks that a corrupted identity is not a prefix of a challenged identity.

Definition 14. *Let* $\mathrm{Adv}^{\text{IND-CPA}}_{\mathsf{HIBE},\mathcal{A}} := 2\Pr[\text{IND-CPA}_{\mathsf{HIBE},\mathcal{A}} = \mathtt{true}] - 1$ *denote the advantage of* $\mathcal{A}$ *against the game defined in Fig. 11 without the Decrypt oracle. A scheme* HIBE *is* IND-CPA *secure, if for all PPT adversaries* $\mathcal{A}$, $\mathrm{Adv}^{\text{IND-CPA}}_{\mathsf{HIBE},\mathcal{A}}$ *is negligible in* κ.

The scheme. The scheme, depicted in Fig. 12, then works analogously for the GGM-based FF-PRG: It simply replaces the PRG expansion of a seed into its left and right sub-seeds by two key derivations for the sub-identities 0 and 1.

Encryption works by encrypting to the respective epoch's leaf identity. To this end, we define $\mathrm{binary}(n, \mathsf{h})$ to return the h digit binary representation of n. (Note that epoch number starts at 1, while the corresponding leaf index is the all-zero string.)

Analysis. Analogous to the GGM-based FF-PRG, we can summarize the construction using the following lemma. The proof is analogous to the one of Lemma 1, where instead of PRG security one uses the correctness and IND-CPA security of the HIBE scheme to argue the respective properties.

Lemma 8. *The scheme from Fig. 12 is a correct and IND-CPA secure FF-UPKE, according to Definitions 6 and 7. If* h *denotes the height of the tree (i.e., the construction allows for at most* 2^{h} *epochs), then the secret key consists of* $\mathcal{O}(\log(\mathsf{h}))$ *secret keys of the underlying HIBE scheme, and sequential updating as well as fast-forwarding from epoch* i *to* $j > i$ *take* $\mathcal{O}(\log(\mathsf{h}))$ *secret-key derivations.*

B Homomorphic UPKE Schemes

We rely on standard model constructions of Dodis *et al.* [24] to present the construction of homomorphic UPKE. Their constructions, from DDH and LWE, relied on three properties: combined leakage resilience and circular security, message homomorphism, and key homomorphism. We will leverage the homomorphism properties to show how to build homomorphic UPKE. Due to space constraints, we will not reproduce the security proof of these constructions in full. We invite the reader to refer to the work of Dodis *et al.* [24] for the complete details.

Protocol HIBE based FF-UPKE

The parameter h specifies the depth of the HIBE tree, i.e., the stream cipher supports at most 2^{h} epochs.

$\underline{\mathsf{KeyGen}(1^\kappa)}$
- $(\mathsf{mpk}, \mathsf{hsk}) \leftarrow \mathsf{HIBE.setup}(1^\kappa)$
- $\mathrm{SK}[\cdot] \leftarrow \perp$
- // Expand leftmost path
- **for** $i \leftarrow 1, \ldots, \mathsf{h}$ **do**
  - $\mathsf{hsk}_l \leftarrow \mathsf{HIBE.kg}(\mathsf{hsk}, 0)$
  - $\mathsf{hsk}_r \leftarrow \mathsf{HIBE.kg}(\mathsf{hsk}, 1)$
  - // Keep co-path nodes in state
  - $\mathsf{idx}_r \leftarrow \mathrm{rightChild}(i)$
  - $\mathrm{SK}[\mathsf{idx}_r] \leftarrow \mathsf{hsk}_r$
  - $\mathsf{hsk} \leftarrow \mathsf{hsk}_l$
- $\mathsf{sk}_1 \leftarrow (1, \mathsf{hsk}, \mathrm{SK})$
- $\mathsf{pk}_1 \leftarrow (1, \mathsf{mpk})$
- **return** $(\mathsf{pk}_1, \mathsf{sk}_1, \varnothing)$

$\underline{\mathsf{Encrypt}(\mathsf{pk}_i, \mathsf{m})}$
- **parse** $(i, \mathsf{mpk}) \leftarrow \mathsf{pk}_i$
- $\vec{\mathsf{id}} \leftarrow \mathrm{binary}(i-1, \mathsf{h})$
- $\mathsf{c} \leftarrow \mathsf{HIBE.enc}(\mathsf{mpk}, \vec{\mathsf{id}}, \mathsf{m})$
- **return** c

$\underline{\mathsf{Decrypt}(\mathsf{sk}_i, \mathsf{c})}$
- **parse** $(i, \mathsf{hsk}, \cdot) \leftarrow \mathsf{sk}_i$
- $\mathsf{m} \leftarrow \mathsf{HIBE.dec}(\mathsf{hsk}, \mathsf{c})$
- **return** m

$\underline{\mathsf{UpdatePK}(\mathsf{pk}_i, \mathsf{BB}_{\mathrm{I}_i})}$
- **parse** $(i, \mathsf{mpk}) \leftarrow \mathsf{pk}_i$
- $\mathsf{pk}_{i+1} \leftarrow (i+1, \mathsf{mpk})$
- **return** $(\mathsf{pk}_{i+1}, \varnothing)$

$\underline{\mathsf{UpdatePK\text{-}Idx}(i)}$
- **return** $\varnothing$

$\underline{\mathsf{UpdateSK}(\mathsf{sk}_i, \mathsf{BB}_{\mathrm{I}_i})}$
- **parse** $(i, \mathsf{hsk}, \mathrm{SK}) \leftarrow \mathsf{st}_i$
- **return** $\mathsf{Leap}(\mathsf{sk}_i, i+1, \mathsf{BB}_{\mathrm{I}_i})$

$\underline{\mathsf{UpdateSK\text{-}Idx}(i)}$
- **return** $\varnothing$

$\underline{\mathsf{LeapSK}(\mathsf{sk}_i, j, \mathsf{BB}_{\mathrm{I}_{i,j}})}$
- **parse** $(i, \cdot, \mathrm{SK}) \leftarrow \mathsf{sk}_i$
- $\mathsf{idx} \leftarrow \mathsf{leaf}(i)$
- $\mathsf{idx}' \leftarrow \mathsf{leaf}(j)$
- // Find copath ancestor of idx'
- **repeat**
  - $\mathsf{idx} \leftarrow \mathsf{rCoPath}(\mathsf{idx})$
  - $\mathsf{hsk} \leftarrow \mathrm{SK}[\mathsf{idx}]$
  - $\mathrm{SK}[\mathsf{idx}] \leftarrow \perp$ // Delete for forward secrecy
  - $\mathsf{idx} \leftarrow \mathsf{idx}'$
- **until** $\mathrm{isAncestor}(\mathsf{idx}, \mathsf{idx}')$
- // Derive path to idx'
- **while** $\mathsf{idx} \neq \mathsf{idx}'$ **do**
  - **if** $\mathrm{isAncestor}(\mathsf{idx}+1, \mathsf{idx}')$ **then** // Go down left
    - $\mathsf{hsk}_l \leftarrow \mathsf{HIBE.kg}(\mathsf{hsk}, 0)$
    - $\mathsf{hsk}_r \leftarrow \mathsf{HIBE.kg}(\mathsf{hsk}, 1)$
    - $\mathsf{idx}_r \leftarrow \mathrm{rightChild}(\mathsf{idx})$
    - $\mathrm{SK}[\mathsf{idx}_r] \leftarrow \mathsf{hsk}_r$ // Store right copath
    - $\mathsf{idx} \leftarrow \mathsf{idx}+1$
    - $\mathsf{hsk} \leftarrow \mathsf{hsk}_l$
  - **else** // Go down right
    - $\mathsf{hsk} \leftarrow \mathsf{HIBE.kg}(\mathsf{hsk}, 1)$
    - $\mathsf{idx} \leftarrow \mathrm{rightChild}(\mathsf{idx})$
- $\mathsf{sk}_j \leftarrow (j, \mathsf{hsk}, \mathrm{SK})$
- **return** sk_j

$\underline{\mathsf{LeapSK\text{-}Idx}(i, j)}$
- **return** $\varnothing$

Fig. 12: The HIBE-based FF-UPKE.

B.1 An LWE-Based Construction

We now re-frame their LWE Construction according to our syntax. There are some critical differences between their construction and ours. First, we increase the message space from bits to $\mathbb{Z}_q$ with $q \ll p$. In other words, rather than encoding the input m as an element in $\mathbb{Z}_2$, we encode it as $\mathbb{Z}_q$. Further, the $\star$ operation is also defined over this space $\mathbb{Z}_q$. We will then prove that this construction is indeed update-homomorphic. This construction is presented as Fig. 13. Finally, as a corollary of the results of Brakerski and Vaikuntanathan [17], we get the following theorem:

Theorem 9. *Under the LWE assumption, the construction presented in Fig. 13 is update-homomorphic, i.e., a correct and secure* H-UPKE *scheme, supporting the aggregation of up to q atomic updates.*

B.2 A DDH-Based Construction

We now present their DDH Construction but in our definition. Note that in this construction update ciphertext has a value z at the beginning. This is to represent the upper bound on the value each element of $\boldsymbol{\delta}$. In other words, encrypting $\boldsymbol{\delta}$ that is a bit string, will have the value of z as 1. However, when one combines updates corresponding to two values $\boldsymbol{\delta}, \boldsymbol{\delta}'$ where each has an upper bound z and z', it produces a resulting encryption of $\boldsymbol{\delta}''$ with upper bound $z + z'$. This can be seen in the function Upd-Comb.

While this was observed in the LWE-based construction as well, this construction critically uses it to efficiently compute the discrete logarithm. As a result, using for instance Pollard's rho algorithm, fast-forwarding from epoch i to j ends up having local computation time $\mathcal{O}(\sqrt{j-i})$ instead of $\mathcal{O}(\log(j))$.

Theorem 10. *Under the DDH Assumption, the construction as presented in Fig. 14 is update-homomorphic.*

Proof. This proof follows from the definition and correctness of the underlying UPKE Scheme.

Protocol LWE-Based UPKE

$\underline{\mathsf{KeyGen}(1^\kappa)}$

Sample $\boldsymbol{A} \leftarrow\$ \mathbb{Z}_p^{n\times m}$
Sample $\boldsymbol{s}_1 \leftarrow\$ \{0,1\}^m$
Compute $\boldsymbol{u}_1 = \boldsymbol{A}\boldsymbol{s}$
return $(\mathsf{pk}_1 = (\boldsymbol{A}, \boldsymbol{u}_1), \mathsf{sk}_1 = (\boldsymbol{s}_1))$

$\underline{\mathsf{Encrypt}(\mathsf{pk}_i, b \in \mathbb{Z}_q)}$

Parse $\mathsf{pk}_i = (\boldsymbol{A}, \boldsymbol{u}_i)$
Sample $\boldsymbol{x} \leftarrow\$ \mathbb{Z}_p^n, \boldsymbol{e} \leftarrow\$ \chi^m, e' \leftarrow \chi'$
Compute $\boldsymbol{t} = \boldsymbol{A}^T\boldsymbol{x} + \boldsymbol{e}$, $pad = \langle \boldsymbol{x}, \boldsymbol{u}\rangle + e' + b\lfloor p/q \rfloor$
return $c = (\boldsymbol{t}, pad)$

$\underline{\mathsf{Decrypt}(\mathsf{sk}_i, c)}$

Parse $c = (\boldsymbol{t}, pad)$ and $\mathsf{sk}_i = \boldsymbol{s}_i$
Compute $b' = (pad - \langle \boldsymbol{s}_i, \boldsymbol{t}\rangle) \in \mathbb{Z}_p$
return $\lfloor b' \cdot q/p \rfloor$

$\underline{\mathsf{UpdGen}(1^\kappa)}$

Sample $\boldsymbol{\delta}_i = (\delta_1, \ldots, \delta_m) \leftarrow\$ \{0,1\}^m$
return $\boldsymbol{\delta}_i$

$\underline{\mathsf{UpdEnc}(\mathsf{pk}_i, \boldsymbol{\delta}_{i+1})}$

Encrypt $\boldsymbol{\delta}$ bit-by-bit, i.e., $\mathsf{up} = (1, (\mathsf{Encrypt}(\mathsf{pk}_i, \delta_1), \ldots, \mathsf{Encrypt}(\mathsf{pk}_i, \delta_m)))$.
return up

$\underline{\mathsf{UpdDec}(\mathsf{sk}_i, \mathsf{up}_{i+1})}$

Parse $\mathsf{up} = (z, c_1, \ldots, c_m)$
for $j = 1, \ldots, m$ **do**
  Compute $\delta_j = \mathsf{Decrypt}(\mathsf{sk}_i, c_j)$
return $\boldsymbol{\delta} = (\delta_1, \ldots, \delta_m)$

$\underline{\mathsf{UpdatePK}(\mathsf{pk}_i, \boldsymbol{\delta}_{i+1})}$

Parse $\mathsf{pk}_i = (\boldsymbol{A}, \boldsymbol{u}_i)$
Compute $\boldsymbol{u}_{i+1} = \boldsymbol{u} + \boldsymbol{A}\boldsymbol{\delta}$
return $\mathsf{pk}_{i+1} = (\boldsymbol{A}, \boldsymbol{u}_{i+1})$

$\underline{\mathsf{Upd\text{-}Comb}(\mathsf{up}_1, \mathsf{up}_2)}$

Parse $\mathsf{up}_1 = (z, c_1, \ldots, c_m)$, $\mathsf{up}_2 = (z', c_1', \ldots, c_m')$
for $j = 1, \ldots, m$ **do**
  Parse $c_j = (\boldsymbol{t}_j, pad_j)$ and $c_j' = (\boldsymbol{t}_j', pad_j')$
  Compute $c_j'' = (\boldsymbol{t}_j + \boldsymbol{t}_j', pad_j + pad_j')$
return $\mathsf{up} = (z + z', c_1'', \ldots, c_m'')$

Fig. 13: LWE Based Construction. Let n, m, p, q be integer parameters of the scheme. We will assume that LWE holds where q is super-polynomial, $p \gg q$ (p defined as product of q and another super-polynomial) and χ is polynomially bounded. Then, we set χ' to be uniformly random over (say) $\left[\frac{-p}{4q}, \frac{p}{4q}\right]$. Further, we have that $m \geq \frac{(n+1)}{\log_2(4/3)} \log p + \omega(\log \kappa)$. Finally, in this construction $\star$ is element-by-element addition over $\mathbb{Z}_q$.

Protocol DDH-Based UPKE

$\underline{\mathsf{KeyGen}(1^\kappa)}$

$\mathbb{G}, g \leftarrow\$ \mathcal{G}(1^\kappa)$
Sample $\boldsymbol{s}_1 = (s_1, \ldots, s_\ell) \leftarrow\$ \{0,1\}$ and $g_1, \ldots, g_\ell \leftarrow\$ \mathbb{G}$.
Compute $h = \prod_{j=1}^{\ell} g_j^{s_j}$.
return $\mathsf{sk}_1 = \boldsymbol{s}_1 \in \mathbb{Z}_p^\ell$, $\mathsf{pk}_1 = (g_1, \ldots, g_\ell, h) \in \mathbb{G}^{\ell+1}$, $\mathsf{pp} = \mathbb{G}$

$\underline{\mathsf{Encrypt}(\mathsf{pk}_i, m \in \mathbb{G})}$

Parse $\mathsf{pk}_i = (g_1, \ldots, g_\ell, h)$
Sample $r \leftarrow\$ \mathbb{Z}_p$
for $j = 1, \ldots, \ell$ **do**
  Compute $f_j = g_j^r$
return $C = (f_1, \ldots, f_\ell, c = h^r \cdot m) \in \mathbb{G}^{\ell+1}$

$\underline{\mathsf{Decrypt}(\mathsf{sk}_i, C)}$

Parse $C = (f_1, \ldots, f_\ell, c = h^r \cdot m)$ and $\mathsf{sk}_i = \boldsymbol{s}_i = (s_1, \ldots, s_\ell) \in \mathbb{Z}_p^\ell$
Compute $m' = c \cdot \left(\prod_{j=1}^{\ell} f_j^{s_j}\right)^{-1}$
return m'

$\underline{\mathsf{UpdGen}(\mathsf{pp})}$

Sample $\boldsymbol{\delta}_i = (\delta_1, \ldots, \delta_m) \leftarrow\$ \{0,1\}^\ell$
return $\boldsymbol{\delta}_i$

$\underline{\mathsf{UpdEnc}(\mathsf{pk}_i, \boldsymbol{\delta}_{i+1})}$

Encrypt $\boldsymbol{\delta}$ bit-by-bit, i.e., $\mathsf{up} = \left(1, \left(\mathsf{Encrypt}(\mathsf{pk}_i, g^{\delta_1}), \ldots, \mathsf{Encrypt}(\mathsf{pk}_i, g^{\delta_\ell})\right)\right)$.
return up

$\underline{\mathsf{UpdDec}(\mathsf{sk}_i, \mathsf{up}_{i+1})}$

Parse $\mathsf{up} = (z, c_1, \ldots, c_\ell)$
for $j = 1, \ldots, \ell$ **do**
  Compute $u_j = \mathsf{Decrypt}(\mathsf{sk}, c_j)$
  Compute δ_j from $u_j = g^{\delta_j}$ where $\delta_j \in \{0, \ldots, z\}$
return $\boldsymbol{\delta} = (\delta_1, \ldots, \delta_\ell)$

$\underline{\mathsf{UpdatePK}(\mathsf{pk}_i, \boldsymbol{\delta}_{i+1})}$

Parse $\mathsf{pk}_i = (g_1, \ldots, g_\ell, h)$ and $\boldsymbol{\delta}_{i+1} = (\delta_1, \ldots, \delta_\ell)$
Compute $h' = h \cdot \left(\prod_{j=1}^{\ell} g_j^{\delta_j}\right)$
return $\mathsf{pk}_{i+1} = (g_1, \ldots, g_\ell, h')$

$\underline{\mathsf{Upd\text{-}Comb}(\mathsf{up}_1, \mathsf{up}_2)}$

Parse $\mathsf{up}_1 = (z, c_1, \ldots, c_\ell)$, $\mathsf{up}_2 = (z', c_1', \ldots, c_\ell')$
for $j = 1, \ldots, \ell$ **do**
  Parse $c_j = (f_1, \ldots, f_\ell, c)$ and $c_j' = (f_1', \ldots, f_\ell', c')$
  Compute $c_j'' = (f_1 \cdot f_1', \ldots, f_\ell \cdot f_\ell', c \cdot c')$
return $\mathsf{up} = (z + z', c_1'', \ldots, c_\ell'')$

Fig. 14: DDH Based H-UPKE Construction. Let $\mathcal{G}$ be a probabilistic polynomial-time "group generator" that takes as input 1^κ and outputs the description of a group $\mathbb{G}$ with prime order $p = p(\kappa)$ and g is a fixed generator of $\mathbb{G}$. Set $\ell = \lceil 5 \log p \rceil$. Finally, in this construction $\star$ is element-by-element addition over $\mathbb{Z}_p$.

C Details on the Fast-Forwardable PRG Construction

In this section, we provide a formal description of the (basic) fast-forwardable PRG construction in the bulletin-board model.

To this end, we consider the construction for up to 2^{h} epochs that makes use of a GGM tree of height h. We assume that the tree nodes are index according to the *preorder* depth-first traversal of the tree. Observe that this implies that for an inner node idx, its left child has an index $\mathsf{idx}+1$. Moreover, if idx is just one level above the leaves (i.e, if its children are leaves) then its right child has index $\mathsf{idx}+2$. To simplify the exposition, we additionally introduce the following notation:

rightChild(idx)	Returns the right child's index of the inner node idx.
isAncestor($\mathsf{idx}, \mathsf{idx}'$)	Returns true iff idx is an ancestor of idx', i.e., iff idx lies on the path from idx' to the root.
$\mathsf{leaf}(e)$	Returns the e-th leaf. (Corresponding to the e-th epoch).
isLeaf(idx)	Returns true iff idx denotes a leaf node.
$\mathsf{prevLeaf}(\mathsf{idx})$	Returns the next lower index $\mathsf{idx}' < \mathsf{idx}$ such that isLeaf(idx'). (Undefined if no such node exists.)
nextLeaf(idx)	Returns the next higher index $\mathsf{idx}' > \mathsf{idx}$ such that isLeaf(idx'). (Undefined if no such node exists.)
$\mathsf{rCoPath}(\mathsf{idx})$	Returns the first node on idx's right copath. (Undefined if no such node exists.)

The algorithms are depicted in Figs. 15 and 16. The Init algorithm takes the GGM tree's seed as input and then proceeds to expand the tree's leftmost path. As part of this process, it sequentially (bottom-up) stores encryption of the copath on the bulletin board. That is, if v_l is a node on the leftmost path, then denote by u_l and u_r its left and right children, respectively, and by v_r its sibling node. Then, the algorithm stores the encryption of the copath node v_r under the key of the next copath node u_r. Additionally, the algorithm stores encryption of the last copath node (the leaf) under its left sibling's key v. The Init algorithm outputs v's seed s_v as the first random value, and stores its index idx_v and key k_v as part of the state. Moreover, it keeps indices, seeds, and keys of the last three copath nodes v_{f^-}, v_{f}, and v_{f} — which represent the frontier at this point — as part of the state.

The Update algorithm moves to the next leaf by fetching from the bulletin board the ciphertext that encrypts its seed and key under the current leaf's key. Recall that this key is stored as part of the local state. The algorithm then expands the next two internal nodes by expanding the frontier node twice. For each such step, v_{f} is used to derive the seeds and keys of the respective child nodes v_l and v_r and the algorithm then uploads the following encryptions to the bulletin board: the node v_l under the last fully expanded node f^-'s key, v_r under v_l's key, and f^+ under v_r's key. If v_l and v_r are leaves, then v_r becomes the last fully expanded node f'^-. Its first right copath node $\mathsf{rCoPath}(\mathsf{idx}_r)$, which is equal to the old f^+, is thus the next node to expand: the new frontier f'. The algorithm now needs to store $\mathsf{rCoPath}(\mathsf{rCoPath}(\mathsf{idx}_r))$ as the new f^+. To this end, it fetches the encryption of that node under $\mathsf{rCoPath}(\mathsf{idx}_r)$ from the bulletin board. If on the other hand v_l is an internal node, then v_l becomes the new frontier f and f^- remains unchanged. Note that in that case $\mathsf{rCoPath}(\mathsf{idx}_r) = \mathsf{rCoPath}(\mathsf{idx}_{\mathsf{f}})$ and thus f^+ remains unchanged as well.

The $\mathsf{Update\text{-}Idx}$ algorithm thus needs to identify the following encryptions: First, the indices $(\mathsf{leaf}(i), \mathsf{leaf}(i+1))$ to decrypt the next leaf. Second, it needs to determine whether an additional link is needed to recover the node v'_{f^+} during the expansion. If so, it additionally outputs the indices $(\mathsf{idx}_{\mathsf{f}^+}, \mathsf{idx}'_{\mathsf{f}^+})$. Note that this can only happen for the former of the two expansions: If the first expansion step results in two leaf nodes, then $v'_{\mathsf{f}} = v_{\mathsf{f}^+}$ and v'_{f^+} needs to be restored from the bulletin board; the second node to be expanded, v'_{f}, on the other hand then is higher up in the tree. If the first expansion step does not result in leaf nodes, then we either get $v''_{\mathsf{f}^+} = \mathrm{rightChild}(v'_{\mathsf{f}})$ or $v''_{\mathsf{f}^+} = v_{\mathsf{f}^+}$. (For simplicity, the pseudo-code description then recovers v''_{f^+} using the link $(\mathsf{idx}'_{\mathsf{f}^+}, \mathsf{idx}''_{\mathsf{f}^+})$ we just produced during the first expansion and hence is in BB_{up}. Of course this could be optimized avoiding the unnecessary encryption and decryption.)

To fast forward from epoch i to j, the Leap algorithm first determines the least-common ancestor of the copath of epoch i and the direct path of epoch j. This is done as part of the recoverNode helper from Fig. 16 that walks up to copath (by fetching the respective encryptions from the bulletin board) until an ancestor of $\mathsf{leaf}(j)$ is hit. From there, it then derives the necessary seeds and keys. The nodes v_{f^-}, v_{f}, and v_{f^+} are recovered analogously. The $\mathsf{Leap\text{-}Idx}$ algorithm thus has to determine all the required encryptions from the bulletin board to recover the required part of epoch i's copath. Observe that v_{f^+} (if it still exists for epoch j) is right of v_{f}, and thus potentially needs more of the copath, which in turn is further to the right of $\mathsf{leaf}(j)$.

Protocol Fast-Forwardable PRG

The parameter h specifies the depth of the GGM tree, i.e., the stream cipher supports at most 2^{h} epochs.

Init(key)

$\mathsf{BB}_{init}[\cdot] \leftarrow \bot$
// Expand leftmost path
$\mathsf{s}_l \leftarrow \mathsf{key}$
for $i \leftarrow 1, \ldots, \mathsf{h}$ **do**
  $(\mathsf{s}_l, \mathsf{k}_l, \mathsf{s}'_r, \mathsf{k}'_r) \leftarrow \mathsf{PRG.Expand}(\mathsf{s}_l)$
  $\mathsf{idx}'_r \leftarrow \mathrm{rightChild}(i)$
  if $i \neq 1$ **then**
    $\mathsf{c} \leftarrow \mathsf{SE.Enc}(\mathsf{k}'_r, (\mathsf{s}_r, \mathsf{k}_r), \mathsf{idx}_r)$
    $\mathsf{BB}_{init}[\mathsf{idx}'_r, \mathsf{idx}_r] \leftarrow \mathsf{c}$
  // Keep 3 last co-path nodes in state
  if $i = \mathsf{h}$ **then**
    $v_{\mathsf{f}-} \leftarrow (\mathsf{idx}'_r, \mathsf{s}'_r, \mathsf{k}'_r)$
  else if $i = \mathsf{h} - 1$ **then**
    $v_{\mathsf{f}} \leftarrow (\mathsf{idx}'_r, \mathsf{s}'_r, \mathsf{k}'_r)$
  else if $i = \mathsf{h} - 2$ **then**
    $v_{\mathsf{f}+} \leftarrow (\mathsf{idx}'_r, \mathsf{s}'_r, \mathsf{k}'_r)$
  $(\mathsf{s}_r, \mathsf{k}_r, \mathsf{idx}_r) \leftarrow (\mathsf{s}'_r, \mathsf{k}'_r, \mathsf{idx}'_r)$
$\mathsf{c} \leftarrow \mathsf{SE.Enc}(\mathsf{k}_l, (\mathsf{s}_r, \mathsf{k}_r), \mathsf{h}+1)$
$\mathsf{BB}_{init}[\mathsf{h}+1, \mathsf{h}+2] \leftarrow \mathsf{c}$
$\mathsf{st}_1 \leftarrow (\mathsf{h}+1, \mathsf{k}_l, v_{\mathsf{f}-}, v_{\mathsf{f}}, v_{\mathsf{f}+})$
$\mathsf{R}_1 \leftarrow \mathsf{s}_l$
return $(\mathsf{st}_1, \mathsf{R}_1, \mathsf{BB}_{init})$

Update-Idx(i)

if $i \geq 2^{\mathsf{h}+1}$ **then**
  return $\varnothing$
// Next leaf
$\mathrm{I}_i \leftarrow \{(\mathsf{leaf}(i), \mathsf{leaf}(i+1))\}$
// Potential change of f^+
$\mathsf{idx}_{\mathsf{f}} \leftarrow \mathrm{indexOfFrontier}(i)$
if $\mathsf{idx}_{\mathsf{f}} \neq \bot \wedge \mathrm{isLeaf}(\mathsf{idx}_{\mathsf{f}} + 1)$ **then**
  $\mathsf{idx}_{\mathsf{f}+} \leftarrow \mathsf{rCoPath}(\mathsf{idx}_{\mathsf{f}})$
  if $\mathsf{idx}_{\mathsf{f}+} \neq \bot$ **then**
    $\mathsf{idx}_{\mathsf{f}++} \leftarrow \mathsf{rCoPath}(\mathsf{idx}_{\mathsf{f}+})$
    if $\mathsf{idx}_{\mathsf{f}++} \neq \bot$ **then**
      $\mathrm{I}_i \leftarrow \mathrm{I}_i \cup \{(\mathsf{idx}_{\mathsf{f}+}, \mathsf{idx}_{\mathsf{f}++})\}$
return I_i

Leap-Idx(i, j)

$\mathrm{I}_{i,j} \leftarrow \varnothing$
$\mathsf{idx} \leftarrow \mathsf{leaf}(i)$
$\mathsf{idx}_t \leftarrow \mathsf{leaf}(j)$
$\mathsf{f} \leftarrow \mathrm{indexOfFrontier}(j)$
if $\mathsf{f} \neq \bot$ **then**
  $\mathsf{f}^+ \leftarrow \mathsf{rCoPath}(\mathsf{f})$
  if $\mathsf{f}^+ \neq \bot$ **then**
    $\mathsf{idx}_t \leftarrow \mathsf{f}^+$
  else
    $\mathsf{idx}_t \leftarrow \mathsf{f}$
repeat
  $\mathsf{idx}' \leftarrow \mathsf{rCoPath}(\mathsf{idx})$
  $\mathrm{I}_{i,j} \leftarrow \mathrm{I}_{i,j} \cup \{(\mathsf{idx}, \mathsf{idx}')\}$
  $\mathsf{idx} \leftarrow \mathsf{idx}'$
until $\mathrm{isAncestor}(\mathsf{idx}, \mathsf{idx}_t)$

Update($\mathsf{st}_i, \mathsf{BB}_{\mathrm{I}_i}$)

parse $(\mathsf{idx}, \mathsf{k}_{\mathsf{idx}}, v_{\mathsf{f}-}, v_{\mathsf{f}}, v_{\mathsf{f}+}) \leftarrow \mathsf{st}_i$
// Decrypt output
$\mathsf{idx}' \leftarrow \mathrm{nextLeaf}(\mathsf{idx})$
$\mathsf{c} \leftarrow \mathsf{BB}_{\mathrm{I}_i}[\mathsf{idx}, \mathsf{idx}']$
$(\mathsf{s}_{\mathsf{idx}'}, \mathsf{k}_{\mathsf{idx}'}) \leftarrow \mathsf{SE.Dec}(\mathsf{k}_{\mathsf{idx}}, \mathsf{c}, \mathsf{idx}')$
$\mathsf{R}_{i+1} \leftarrow \mathsf{s}_{\mathsf{idx}'}$
// Expand (up to) two internal nodes
if $v_{\mathsf{f}} \neq \bot$ **then**
  $(v'_{\mathsf{f}-}, v'_{\mathsf{f}}, v'_{\mathsf{f}+}, \mathsf{BB}_{up}) \leftarrow \mathrm{expandFrontier}(v_{\mathsf{f}-}, v_{\mathsf{f}}, v_{\mathsf{f}+}, \mathsf{BB}_{\mathrm{I}_i})$
  if $v'_{\mathsf{f}} \neq \bot$ **then**
    $(v'_{\mathsf{f}-}, v'_{\mathsf{f}}, v'_{\mathsf{f}+}, \mathsf{BB}'_{up}) \leftarrow \mathrm{expandFrontier}(v'_{\mathsf{f}-}, v'_{\mathsf{f}}, v'_{\mathsf{f}+}, \mathsf{BB}_{\mathrm{I}_i} \cup \mathsf{BB}_{up})$
    $\mathsf{BB}_{up} \leftarrow \mathsf{BB}_{up} \cup \mathsf{BB}'_{up}$ // merge (disjoint) BBs
$\mathsf{st}_{i+1} \leftarrow (\mathsf{idx}', \mathsf{k}_{\mathsf{idx}'}, v'_{\mathsf{f}-}, v'_{\mathsf{f}}, v'_{\mathsf{f}+})$
return $(\mathsf{st}_{i+1}, \mathsf{R}_{i+1}, \mathsf{BB}_{up})$

Leap($\mathsf{st}_i, j, \mathsf{BB}_{\mathrm{I}_{i,j}}$)

// Derive current seed and key
$\mathsf{idx} \leftarrow \mathsf{leaf}(j)$
$(\mathsf{s}, \mathsf{k}) \leftarrow \mathrm{recoverNode}(\mathsf{st}_i, \mathsf{BB}_{\mathrm{I}_{i,j}}, \mathsf{idx})$
// Derive frontier state
$(v_{\mathsf{f}-}, v_{\mathsf{f}}, v_{\mathsf{f}+}) \leftarrow \bot$
$\mathsf{idx}_{\mathsf{f}} \leftarrow \mathrm{indexOfFrontier}(j)$
if $\mathsf{idx}_{\mathsf{f}} \neq \bot$ **then**
  $(\mathsf{s}_{\mathsf{f}}, \mathsf{k}_{\mathsf{f}}) \leftarrow \mathrm{recoverNode}(\mathsf{st}_i, \mathsf{BB}_{\mathrm{I}_{i,j}}, \mathsf{idx}_{\mathsf{f}})$
  $v_{\mathsf{f}} \leftarrow (\mathsf{idx}_{\mathsf{f}}, \mathsf{s}_{\mathsf{f}}, \mathsf{k}_{\mathsf{f}})$
  $\mathsf{idx}_{\mathsf{f}-} \leftarrow \mathsf{prevLeaf}(\mathsf{f})$
  $(\mathsf{s}_{\mathsf{f}-}, \mathsf{k}_{\mathsf{f}-}) \leftarrow \mathrm{recoverNode}(\mathsf{st}_i, \mathsf{BB}_{\mathrm{I}_{i,j}}, \mathsf{idx}_{\mathsf{f}-})$
  $v_{\mathsf{f}-} \leftarrow (\mathsf{idx}_{\mathsf{f}-}, \mathsf{s}_{\mathsf{f}-}, \mathsf{k}_{\mathsf{f}-})$
  $\mathsf{idx}_{\mathsf{f}+} \leftarrow \mathsf{rCoPath}(\mathsf{f})$
  if $\mathsf{idx}_{\mathsf{f}+} \neq \bot$ **then**
    $(\mathsf{s}_{\mathsf{f}+}, \mathsf{k}_{\mathsf{f}+}) \leftarrow \mathrm{recoverNode}(\mathsf{st}_i, \mathsf{BB}_{\mathrm{I}_{i,j}}, \mathsf{idx}_{\mathsf{f}+})$
    $v_{\mathsf{f}+} \leftarrow (\mathsf{idx}_{\mathsf{f}+}, \mathsf{s}_{\mathsf{f}+}, \mathsf{k}_{\mathsf{f}+})$
// Assemble new state
$\mathsf{st}_j \leftarrow (\mathsf{idx}, \mathsf{k}, v_{\mathsf{f}-}, v_{\mathsf{f}}, v_{\mathsf{f}+})$
return $(\mathsf{st}_j, \mathsf{s})$

Fig. 15: A formal description of the FF Stream Cipher protocol.

Protocol FF Stream-Cipher Helpers

indexOfFrontier(e)

// The frontier associated with epoch e
$\mathsf{idx} \leftarrow \mathsf{h} - 1$
for $j \leftarrow 1, \ldots, 2e$ **do**
  if $\mathsf{idx} \geq 2^{\mathsf{h}+1} - 1$ **then**
    return $\perp$
  else if isLeaf($\mathsf{idx} + 1$) **then**
    $\mathsf{idx} \leftarrow \mathsf{rCoPath}(\mathsf{idx})$
  else
    $\mathsf{idx} \leftarrow \mathsf{idx} + 1$
return idx

recoverNode($\mathsf{st}, \mathsf{BB}, \mathsf{idx}_t$)

// Derives seed and key of node idx_t
parse $(\mathsf{idx}, \mathsf{k}, \cdot, \cdot, \cdot) \leftarrow \mathsf{st}$
repeat
  $\mathsf{idx}' \leftarrow \mathsf{rCoPath}(\mathsf{idx})$
  $\mathsf{c} \leftarrow \mathsf{BB}[\mathsf{idx}, \mathsf{idx}']$
  $(\mathsf{s}_{\mathsf{idx}}, \mathsf{k}_{\mathsf{idx}}) \leftarrow \mathsf{SE.Dec}(\mathsf{k}_{\mathsf{idx}}, \mathsf{c}, \mathsf{idx}')$
  $\mathsf{idx} \leftarrow \mathsf{idx}'$
until isAncestor($\mathsf{idx}, \mathsf{idx}_t$)
while $\mathsf{idx} \neq \mathsf{idx}_t$ **do**
  if isAncestor($\mathsf{idx} + 1, \mathsf{idx}_t$) **then**
    $(\mathsf{s}_{\mathsf{idx}}, \mathsf{k}_{\mathsf{idx}}, \cdot, \cdot) \leftarrow \mathsf{PRG.Expand}(\mathsf{s}_{\mathsf{idx}})$
    $\mathsf{idx} \leftarrow \mathsf{idx} + 1$
  else
    $(\cdot, \cdot, \mathsf{s}_{\mathsf{idx}}, \mathsf{k}_{\mathsf{idx}}) \leftarrow \mathsf{PRG.Expand}(\mathsf{s}_{\mathsf{idx}})$
    $\mathsf{idx} \leftarrow \mathsf{rightChild}(\mathsf{idx})$
return $(\mathsf{s}_{\mathsf{idx}}, \mathsf{k}_{\mathsf{idx}})$

expandFrontier($v_{\mathsf{f}-}, v_{\mathsf{f}}, v_{\mathsf{f}+}, \mathsf{BB}_{\mathrm{I}_i}$)

$\mathsf{BB}_{up}[\cdot] \leftarrow \perp$
parse $(\mathsf{idx}_{\mathsf{f}}, \mathsf{s}_{\mathsf{f}}, \mathsf{k}_{\mathsf{f}}) \leftarrow v_{\mathsf{f}}$
$(\mathsf{s}'_l, \mathsf{k}'_l, \mathsf{s}'_r, \mathsf{k}'_r) \leftarrow \mathsf{PRG.Expand}(\mathsf{s}_{\mathsf{f}})$
$\mathsf{idx}'_l \leftarrow \mathsf{idx}_{\mathsf{f}} + 1$
$\mathsf{idx}'_r \leftarrow \mathrm{rightChild}(\mathsf{idx}_{\mathsf{f}})$
// Upload new links
parse $(\mathsf{idx}_{\mathsf{f}-}, \mathsf{s}_{\mathsf{f}-}, \mathsf{k}_{\mathsf{f}-}) \leftarrow v_{\mathsf{f}-}$
$\mathsf{BB}_{up}[\mathsf{idx}_{\mathsf{f}-}, \mathsf{idx}'_l] \leftarrow \mathsf{SE.Enc}(\mathsf{k}_{\mathsf{f}-}, (\mathsf{s}'_l, \mathsf{k}'_l), \mathsf{idx}'_l)$
$\mathsf{BB}_{up}[\mathsf{idx}'_l, \mathsf{idx}'_r] \leftarrow \mathsf{SE.Enc}(\mathsf{k}'_l, (\mathsf{s}'_r, \mathsf{k}'_r), \mathsf{idx}'_r)$
if $v_{\mathsf{f}+} \neq \perp$ **then**
  parse $(\mathsf{idx}_{\mathsf{f}+}, \mathsf{s}_{\mathsf{f}+}, \mathsf{k}_{\mathsf{f}+}) \leftarrow v_{\mathsf{f}+}$
  $\mathsf{BB}_{up}[\mathsf{idx}'_r, \mathsf{idx}_{\mathsf{f}+}] \leftarrow \mathsf{SE.Enc}(\mathsf{k}'_r, (\mathsf{s}_{\mathsf{f}+}, \mathsf{k}_{\mathsf{f}+}), \mathsf{idx}_{\mathsf{f}+})$
// Adjust frontier
if isLeaf(idx'_l) **then**
  $v'_{\mathsf{f}-} \leftarrow (\mathsf{idx}'_r, \mathsf{s}'_r, \mathsf{k}'_r)$
  $v'_{\mathsf{f}} \leftarrow v_{\mathsf{f}+}$
  $v'_{\mathsf{f}+} \leftarrow \perp$
  if $\mathsf{idx}_{\mathsf{f}+} \neq \perp$ **then**
    $\mathsf{idx}'_{\mathsf{f}+} \leftarrow \mathsf{rCoPath}(\mathsf{idx}_{\mathsf{f}+})$
    if $\mathsf{idx}'_{\mathsf{f}+} \neq \perp$ **then**
      $\mathsf{c} \leftarrow \mathsf{BB}_{\mathrm{I}_i}[\mathsf{idx}_{\mathsf{f}+}, \mathsf{idx}'_{\mathsf{f}+}]$
      $(\mathsf{s}'_{\mathsf{f}+}, \mathsf{k}'_{\mathsf{f}+}) \leftarrow \mathsf{SE.Dec}(\mathsf{k}_{\mathsf{f}+}, \mathsf{c}, \mathsf{idx}'_{\mathsf{f}+})$
      $v'_{\mathsf{f}+} \leftarrow (\mathsf{idx}'_{\mathsf{f}+}, \mathsf{s}'_{\mathsf{f}+}, \mathsf{k}'_{\mathsf{f}+})$
else
  $v'_{\mathsf{f}-} \leftarrow v_{\mathsf{f}-}$
  $v'_{\mathsf{f}} \leftarrow (\mathsf{idx}'_l, \mathsf{s}'_l, \mathsf{k}'_l)$
  $v'_{\mathsf{f}+} \leftarrow (\mathsf{idx}'_r, \mathsf{s}'_r, \mathsf{k}'_r)$
return $(v'_{\mathsf{f}-}, v'_{\mathsf{f}}, v'_{\mathsf{f}+}, \mathsf{BB}_{up})$

Fig. 16: Helper algorithms for the FF Stream Cipher protocol.