Forward-Secure Encryption with Fast Forwarding

Yevgeniy Dodis[1⋆], Daniel Jost[1⋆⋆[0000−0002−6562−9665]], 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 DDH and LWE.

⋆ 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.

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 [32,20] it soon got broadened to incorporate a variety of non-interactive cryptographic primitives, such as forward-secure public-key encryption [16] and forward-secure signatures [6,40]. 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) [7]. 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 [6,40], 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 the full version [21].

Lemma 1 (Informal). *The template [6,40] (for building forward-secure signature schemes) can be adapted to the Goldreich-Goldwasser-Micali (GGM) construction [30] 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?

[3] The number-theoretic Blum-Blum-Shub PRG [8] 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.

Forward-Secure Public-Key Encryption. In the public-key setting, Canetti, Halevi, and Katz [16] 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 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 [9,18,10,26,25,15], 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.[4] 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

[4] 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.

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 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.[5] 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[6] 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) [36,3], 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

[5] These concerns are interesting but orthogonal to our contributions.

[6] Despite some remarkable progress in the construction of pairing-based HIBE (e.g. [10]) over the last decades, those solutions have never gained any widespread adoption, partially for their omission from popular cryptographic libraries.

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, 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. 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 [36,3]. 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 [36] provided a very simple and efficient UPKE (in the random oracle model) that has similar efficiency to the underlying ElGamal encryption. Recently, [22] 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,[7] 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[8] 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 4.

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 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.[9]

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 6 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. [22] both satisfy the above homomorphism.[10]

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

[7] 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.

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

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

[10] Interestingly, the most efficient random-oracle based scheme [36,3] does not appear to be update-homomorphic, and will not be enough for our purposes.

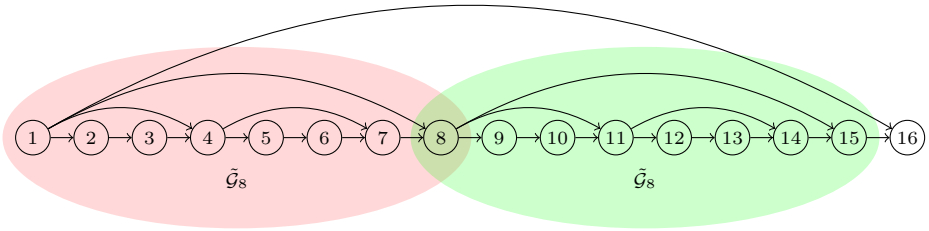

Fig. 1: The update graph $\tilde{\mathcal{G}}_{16}$, consists of two subgraphs $\tilde{\mathcal{G}}_8$ and an extra node.

the collection of edges (i, j) for which parties need to maintain update ciphertexts $\mathsf{up}_{i,j}$. As it turns out (see Definition 8), 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 [42] 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.

1.3 Related Work

Hierarchical Identity Based Encryption. As mentioned before, the work of Canetti, Halevi, and Katz [17] showed how to generically construct Forward-Secure Public Key Encryption from Hierarchical Identity Based Encryption [29,33]. Indeed, over time, various HIBE schemes have been proposed under an assortment of assumptions such as LWE [18,2], CDH/DDH [26,25,15], and pairing-based Diffie-Hellman Assumptions [9,10], 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 [26,25,15], 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 [18,2] rely on either lattice trapdoors or GPV-style pre-image sampling [28] 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. Thus, 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 [4] 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 [6] and later extended and improved by Malkin *et al.* [40]. There are still other constructions that are secure in the random oracle model [1,34,38].

Key Evolving Encryption Schemes. Two recent works - Jaeger and Stepanovs [35] and Poettering and Rössler [41] - 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.* [36] 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.* [22] 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 [12,27,39,37,14,11], vastly different from the idea of UPKE, 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 [23,24], 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 [5,43] and Puncturable PKE [31,19,44] 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 [13]. 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

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.

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 $\left\{\mathsf{PRG.Expand}(\mathsf{s}) \mid \mathsf{s} \leftarrow\!\!\$\ \{0,1\}^\kappa\right\}_{\kappa\in\mathbb{N}}$ and $\left\{\mathsf{k} \leftarrow\!\!\$\ \{0,1\}^{4\kappa}\right\}_{\kappa\in\mathbb{N}}$ 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, $\mathsf{BB}\restriction_{\mathrm{I}} := \{(\mathsf{idx}, v) \in \mathsf{BB} \mid \mathsf{idx} \in \mathrm{I}\}$. Moreover, for bulletin boards BB_1 and BB_2, we define $\mathsf{BB}_1 \mathbin{+\!\!+} \mathsf{BB}_2 := \mathsf{BB}_1 \cup \mathsf{BB}_2\restriction_{\mathcal{I}\setminus\mathrm{dom}(\mathsf{BB}_1)}$, where* dom *denotes the domain.*

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 forward secure and fast forwardable. We, thus, focus on fast-forwardable PRGs[11] instead.

[11] For simplicity, we henceforth omit explicitly mentioning forward secrecy as part of each primitive's name.

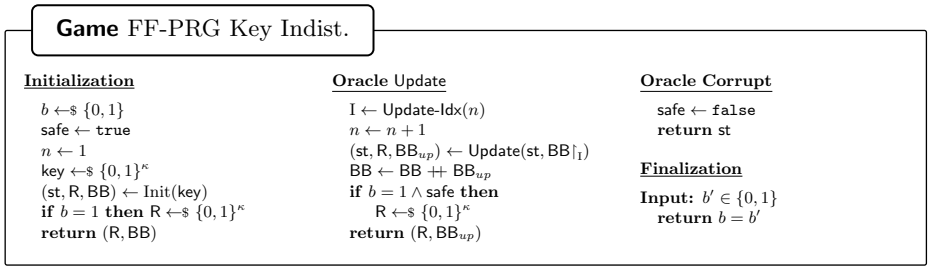

Fig. 2: The security game of FF-PRGs.

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 Leap-Idx to be bounded by fixed polynomials of the security parameter κ, i.e., to be independent of $j - i$. 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. A formal description of the correctness game is presented in the full version [21].

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.

Definition 3 (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 $\mathsf{Update\text{-}Idx}$ and $\mathsf{Leap\text{-}Idx}$ 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 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 4. *A* Fast-Forwardable Updatable Public-Key Encryption scheme *is a tuple of PPT algorithms* $\mathsf{FF\text{-}UPKE} := (\mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Decrypt}, \mathsf{UpdatePK}, \mathsf{UpdatePK\text{-}Idx}, \mathsf{UpdateSK}, \mathsf{UpdateSK\text{-}Idx}, \mathsf{LeapSK}, \mathsf{LeapSK\text{-}Idx})$*, 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 $\mathsf{poly}(\log j)$.

Correctness and Security. 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. A formal definition of correctness can be found in the full version [21].

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 5 (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.*

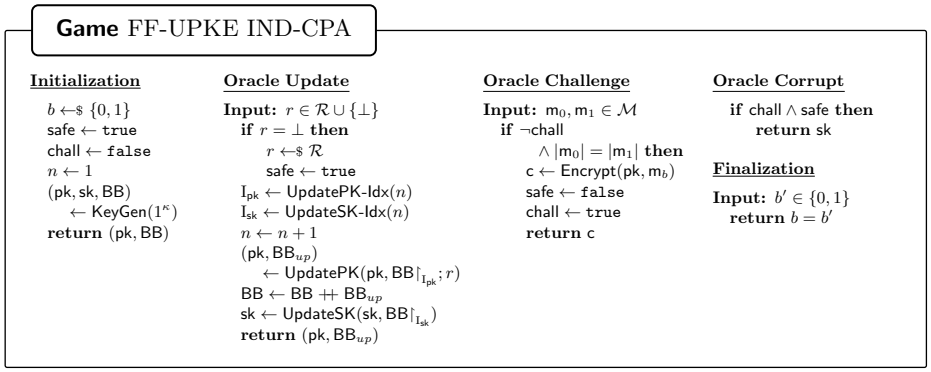

Fig. 3: The IND-CPA game of FF-UPKE.

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 the full version [21] we provide two concrete instantiations of this building block, based off minor modifications the standard-model UPKE schemes introduced in the recent work of Dodis et al. [22]. Both lead two bulletin board values of the order of $\mathcal{O}(\kappa^2)$ many cryptographic elements.

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.

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 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.)

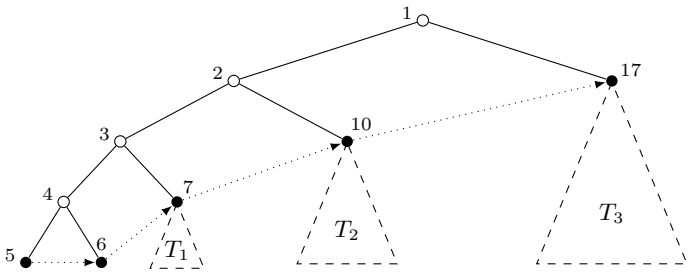

Fig. 4: The tree GGM_4 after the Init algorithm. Dotted arrows represent encryptions outsourced to the bulletin board.

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

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

We observe that by definition $\mathsf{rCoPath}(\text{rightChild}(\mathsf{f})) = \mathsf{rCoPath}(\mathsf{f}) = \mathsf{f}^+$ and $\mathsf{rCoPath}(\text{leftChild}(\mathsf{f})) = \text{rightChild}(\mathsf{f})$. 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., $\mathsf{SE.Enc}(\mathsf{k}_v, (\mathsf{s}_{\mathsf{rCoPath}(v)}, \mathsf{k}_{\mathsf{rCoPath}(v)}), \mathsf{rCoPath}(v))$, 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 leftChild(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 leftChild(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.

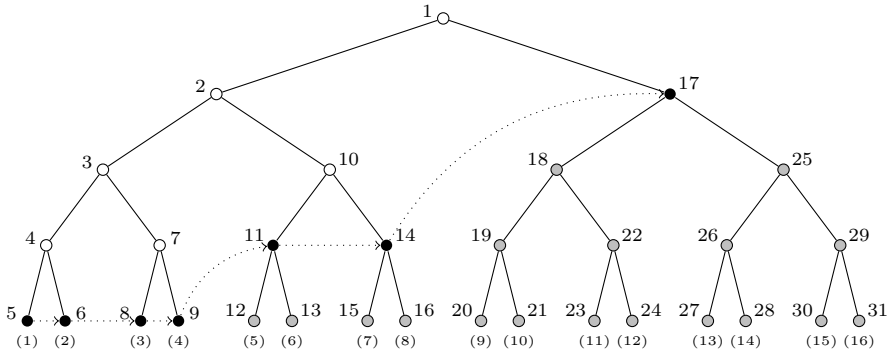

Fig. 5: A visualization of the node expansion in the enhanced GGM construction.

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 $\mathsf{PRG.Expand}$ 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. A proof of the follow theorem is presented in the full version [21].

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.*

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 [40]. Their roots can be derived using a forward-secure PRNG, such as the folklore construction from $\mathsf{PRG.Expand}$. 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.

We refer to the full version of the paper [21] for a more detailed description of the scheme.

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 $\Delta_{[j,\ell]} := (\delta_{j+1} \star \cdots \star \delta_\ell)$, 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 6. *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. 6. 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

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

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. $\delta \leftarrow \mathsf{UpdGen}(\mathsf{pp})$
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. 6.

Definition 7 (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. 6.*

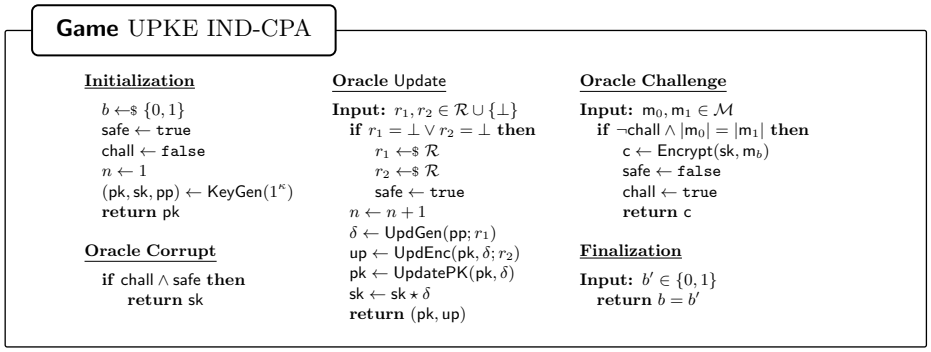

Fig. 6: The IND-CPA game of Update-Homomorphic Updatable Public-Key Encryption (H-UPKE).

Schemes. In the full version [21] we show that with minor modifications the standard-model UPKE schemes of Dodis et al. [22] 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 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 8. *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 9. *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. 7.

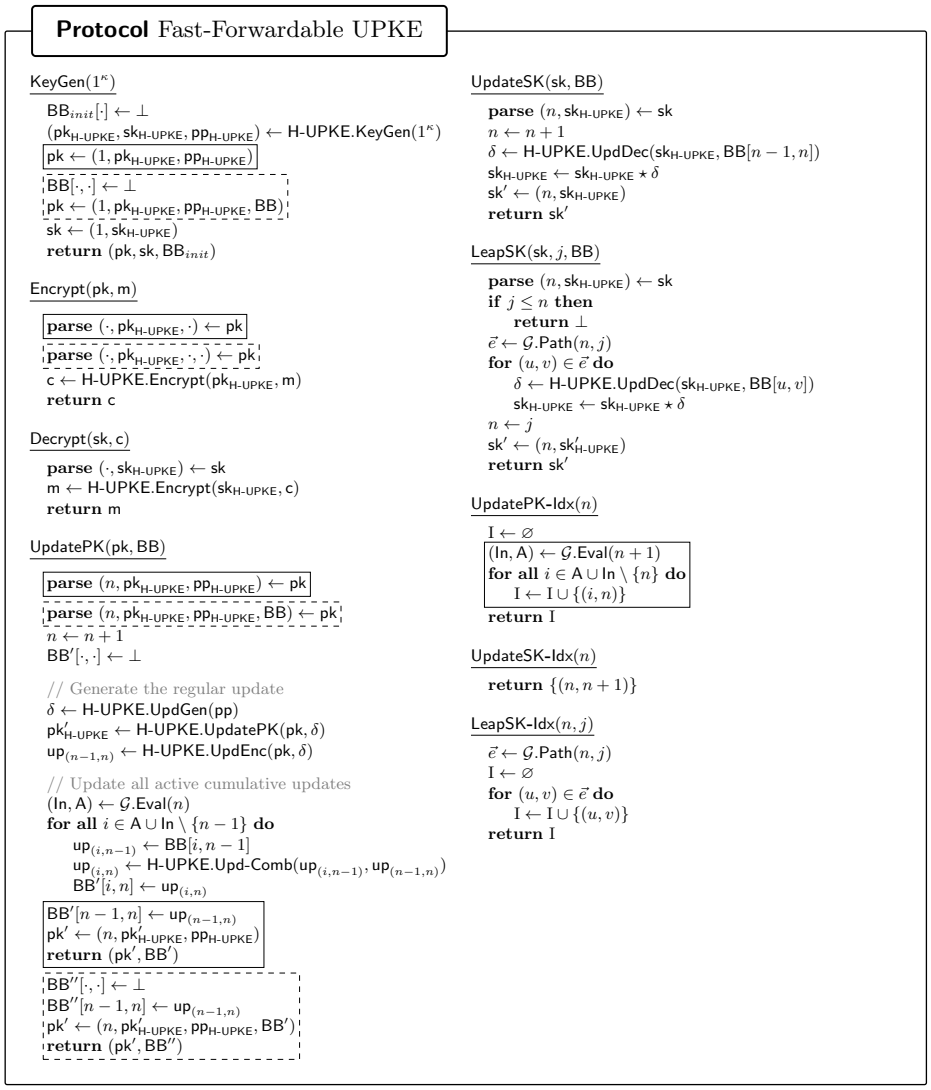

Fig. 7: The FF-UPKE protocols are built from an Update-Homomorphic UPKE scheme H-UPKE and an update graph $\mathcal{G}$. Lines enclosed in solid boxes belong to the regular multi-sender protocol, whereas lines enclosed in dashed boxes represent the single-sender variant.

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. [36]) 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. 7 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 2. *The FF-UPKE schemes presented in Fig. 7 are correct if the underlying scheme* H-UPKE *is correct and update homomorphic as formalized via the games from Fig. 6.*

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 H-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 H-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* $|\mathsf{BB}_{up}| \leq \alpha(n+1)$.

A proof is presented in the full version of the paper [21].

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 3. *The FF-UPKE schemes presented in Fig. 7 are IND-CPA secure, according to Definition 5, if the underlying scheme* H-UPKE *is IND-CPA secure according to Definition 7.*

A proof can be found in the full version [21] of this document.

6 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 [42] 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. Anderson, R.: Invited lecture. Fourth Annual Conference on Computer and Communications Security, ACM (1997)
5. 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
6. 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
7. 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
8. 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)
9. 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
10. 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
11. 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
12. 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
13. 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
14. 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
15. 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
16. 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

17. 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
18. 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
19. 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
20. Diffie, W., van Oorschot, P.C., Wiener, M.J.: Authentication and authenticated key exchanges. Designs, Codes and Cryptography **2**(2), 107–125 (Jun 1992)
21. Dodis, Y., Jost, D., Karthikeyan, H.: Forward-secure encryption with fast forwarding. Cryptology ePrint Archive, Paper 2022/1233 (2022), https://eprint.iacr.org/2022/1233, full version of this report
22. Dodis, Y., Karthikeyan, H., Wichs, D.: Updatable public key encryption in the standard model. In: Theory of Cryptography. Springer International Publishing (2021)
23. 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
24. 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
25. 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
26. 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
27. 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
28. 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
29. 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
30. Goldreich, O., Goldwasser, S., Micali, S.: How to construct random functions. Journal of the ACM **33**(4), 792–807 (Oct 1986)
31. 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

32. 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
33. 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
34. 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
35. 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
36. 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
37. 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
38. 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
39. 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
40. 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
41. 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
42. 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
43. 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
44. 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