Bloom Filter Encryption and Applications to Efficient Forward-Secret 0-RTT Key Exchange

David Derler[1], Tibor Jager[2], Daniel Slamanig[3], and Christoph Striecks[3]

[1] Graz University of Technology
david.derler@tugraz.at
[2] Paderborn University
tibor.jager@upb.de
[3] AIT Austrian Institute of Technology
{daniel.slamanig, christoph.striecks}@ait.ac.at

Abstract. Forward secrecy is considered an essential design goal of modern key establishment (KE) protocols, such as TLS 1.3, for example. Furthermore, efficiency considerations such as zero round-trip time (0-RTT), where a client is able to send cryptographically protected payload data along with the very first KE message, are motivated by the practical demand for secure low-latency communication.

For a long time, it was unclear whether protocols that simultaneously achieve 0-RTT and full forward secrecy exist. Only recently, the first forward-secret 0-RTT protocol was described by Günther et al. (Eurocrypt 2017). It is based on Puncturable Encryption. Forward secrecy is achieved by "puncturing" the secret key after each decryption operation, such that a given ciphertext can only be decrypted once (cf. also Green and Miers, S&P 2015). Unfortunately, their scheme is completely impractical, since one puncturing operation takes between 30 seconds and several minutes for reasonable security and deployment parameters, such that this solution is only a first feasibility result, but not efficient enough to be deployed in practice.

In this paper, we introduce a new primitive that we term Bloom Filter Encryption (BFE), which is derived from the probabilistic Bloom filter data structure. We describe different constructions of BFE schemes, and show how these yield new puncturable encryption mechanisms with extremely efficient puncturing. Most importantly, a puncturing operation only involves a small number of very efficient computations, plus the deletion of certain parts of the secret key, which outperforms previous constructions by orders of magnitude. This gives rise to the first forward-secret 0-RTT protocols that are efficient enough to be deployed in practice. We believe that BFE will find applications beyond forward-secret 0-RTT protocols.

Keywords: Bloom filter encryption ⋄ Bloom filter ⋄ 0-RTT ⋄ forward secrecy ⋄ key exchange ⋄ puncturable encryption

The full version of this paper is available in the IACR Cryptology ePrint archive.

1 Introduction

One central ingredient to secure today's Internet are key exchange (KE) protocols with the most prominent and widely deployed instantiations thereof in the Transport Layer Security (TLS) protocol [15]. Using a KE protocol, two parties (e.g., a server and a client) are able to establish a shared secret (session key) which afterwards can be used to cryptographically protect data to be exchanged between those parties. The process of arriving at a shared secret requires the exchange of messages between client and server, which adds latency overhead to the protocol. The time required to establish a key is usually measured in round-trip times (RTTs). A novel design goal, which was introduced by Google's QUIC protocol and also adopted in the upcoming version of TLS 1.3, aims at developing zero round-trip time (0-RTT) protocols with strong security guarantees. So far, quite some effort was made in the cryptographic literature, e.g. [31,21], and, indeed, 0-RTT protocols are probably going to be used heavily in the future Internet as TLS version 1.3 [28] is approaching fast. Already today, Google's QUIC protocol [29] is used on Google webservers and within the Chrome and Opera browsers to support 0-RTT. Unfortunately, none of the above mentioned protocols are enjoying 0-RTT and full forward secrecy at the same time. Only recently, Günther, Hale, Jager, and Lauer (GHJL henceforth) [20] made progress and proposed the first 0-RTT key exchange protocol with full forward secrecy for all transmitted payload messages. However, although their 0-RTT protocol offers the desired features, their construction is not yet practical.

In more detail, GHJL's forward-secure 0-RTT key-exchange solution is based on puncturable encryption (PE), which they showed can be constructed in a black-box way from any selectively secure hierarchical identity-based encryption (HIBE) scheme. Loosely speaking, PE is a public-key encryption primitive which provides a $\mathsf{Puncture}$ algorithm that, given a secret key and ciphertext, produces an updated secret key that is able to decrypt all ciphertexts except the one it has been punctured on. PE has been introduced by Green and Miers [19] (GM henceforth) who provide an instantiation relying on a binary-tree encryption (BTE) scheme — or selectively secure HIBE — together with a key-policy attribute-based encryption (KP-ABE) [18] scheme for non-monotonic (NM) formulas with specific properties. In particular, the KP-ABE needs to provide a non-standard property to enhance existing secret keys with additional NOT gates, which is satisfied by the NM KP-ABE in [27]. Since then, PE has proved to be a valuable building block to construct public-key watermarking schemes [13], forward-secret proxy re-encryption [14], or for achieving chosen-ciphertext security for fully-homomorphic encryption [11]. However, the mentioned PE instantiations from [11,13] are based on indistinguishability obfuscation and, thus, do not yield practical schemes at all while [14] uses the same techniques as in GHJL.

When looking at the two most efficient PE schemes available, i.e., GM and GHJL, they still come with severe drawbacks. In particular, puncturing in GHJL is highly inefficient and takes several seconds to minutes on decent hardware for reasonable deployment parameters. In the GM scheme, puncturing is more efficient, but the cost of decryption is very significant and increases with the number of puncturings. More precisely, cost of decryption requires a number of pairing evaluations that depends on the number of puncturings, and can be in the order of 2^{10} to 2^{20} for realistic deployment parameters. These issues make both of them especially unsuitable for the application in forward-secret 0-RTT key exchange in a practical setting.

Contributions. In this paper, we introduce Bloom filter encryption (BFE), which can be considered as a variant of PE [19,13,11,20]. The main difference to other existing PE constructions is that in case of BFE, we tolerate a non-negligible correctness error.[4] This allows us to construct PE and in particular puncturable key encapsulation (PKEM) schemes with highly efficient puncturing and in particular where puncturing only requires a few very efficient operations, i.e., to *delete* parts of the secret key, but no further expensive cryptographic operations. Altogether, this makes BFE a very suitable building block to construct practical forward-secret 0-RTT key exchange. In more detail, our contributions are as follows:

- We formalize the notion of BFE by presenting a suitable security model. The intuition behind BFE is to provide a highly efficient decryption and puncturing. Interestingly, puncturing mainly consists of *deleting* parts of the secret key. This approach is in contrast to existing puncturable encryption schemes, where puncturing and/or decryption is a very expensive operation.
- We propose efficient constructions of BFE. First, we present a direct construction which uses ideas from the Boneh-Franklin identity-based encryption (IBE) scheme [9]. Additionally, we present a black-box construction from a ciphertext-policy attibute-based encryption (CP-ABE) scheme that only needs to be small-universe (i.e., bounded) and support threshold policies, which allows us to achieve compact ciphertexts. To improve efficiency, we finally provide a time-based BFE (TB-BFE) from selectively-secure HIBEs.
- To achieve CCA security, we adopt the Fujisaki-Okamoto (FO) transformation [16] to the BFE setting. This is technically non-trivial, and therefore we consider it as another interesting aspect of this work. In particular, the original FO transformation [16] works only for schemes with *perfect* correctness. Recently, Hofheinz et al. [23] described a variant which works also for schemes with *negligible* correctness error. We adopt the FO transformation to BFE and PKEMs with *non-negligible* correctness error respectively. To this end, we formalize additional properties of the PKEM that are required to apply the FO transform to BFE schemes, and show that our CPA-secure constructions satisfy them. This serves as a template that allows an easy application of the FO transform in a black-box manner to BFE schemes.
- We provide a construction of a forward-secret 0-RTT key exchange protocol (in the sense of GHJL) from TB-BFE. Furthermore, we give a detailed comparison of (TB-)BFE with other PE schemes and discuss the efficiency in the context of the proposed application to forward-secret 0-RTT key exchange.

[4] We discuss below why this is not only tolerable, but actually a very reasonable approach for applications like 0-RTT key exchange.

In particular, our construction of forward-secret 0-RTT key-exchange from TB-BFE has none of the drawbacks mentioned in the introduction (at the cost of a somewhat larger secret key, that, however, shrinks with the number of puncturings). Consequently, our forward-secret 0-RTT key exchange can be seen as a significant step forward to construct very *practical* forward-secret 0-RTT key exchange protocols.

On tolerating a non-negligible correctness error for 0-RTT. The huge efficiency gain of our construction stems partially from the relaxation of allowing a non-negligible correctness error, which, in turn, stems from the potentially non-negligible false-positive probability of a Bloom filter. While this is unusual for classical public-key encryption schemes, we consider it as a reasonable approach to accept a small, but non-negligible correctness error for the 0-RTT mode of a key exchange protocol, in exchange for the huge efficiency gain.

For example, a $1/10000$ chance that the key establishment fails allows to use 0-RTT in 9999 out of 10000 cases on average, which is a significant practical efficiency improvement. Furthermore, the communicating parties can implement a fallback mechanism which immediately continues with running a standard 1-RTT key exchange protocol with perfect correctness, if the 0-RTT exchange fails. Thus, the resulting protocol can have the same worst-case efficiency as a 1-RTT protocol, while most of the time 0-RTT is already sufficient to establish a key and full forward secrecy is *always* achieved.

Compared to other practical 0-RTT solutions, note that both TLS 1.3 [28] and QUIC [29] have similar fallback mechanisms. Furthermore, in order to achieve at least a very weak form of forward secrecy, they define so called *tickets* [28] or *server configuration (SCFG)* messages [29], which expire after a certain time. Forward secrecy is only achieved after the ticket/SCFG message has expired and the associated secrets have been erased. Therefore the lifetime should be kept short. If a client connects to a server after the ticket/SCFG message has expired, then the fallback mechanism is invoked and a full 1-RTT handshake is performed. In particular, for settings where a client connects only occasionally to a server, and for reasonably chosen parameters and a moderate life time of the ticket/SCFG message, which at least guarantees some weak form of forward secrecy, this requires a full handshake more often than with our approach.

Finally, note that puncturable encryption with perfect (or negligible) correctness error inherently seems to require secret keys whose size at least grows linearly with the number of puncturings. This is because any such scheme inherently must (implicitly or explicitly) encode information about the list of punctured ciphertexts into the secret key, which lower-bounds the size of the secret key. By tolerating a non-negligible correctness error, we are also able to restrict the growth of the secret key to a limit which seems tolerable in practice.

2 Bloom Filter Encryption

The key idea behind Bloom Filter Encryption (BFE) is that the key pair of such a scheme is associated to a Bloom filter (BF) [7], a probabilistic data structure for the approximate set membership problem with a non-negligible false-positive probability in answering membership queries. The initial secret key sk output by the key generation algorithm of a BFE scheme corresponds to an empty BF where all bits are set to 0. Encryption takes a message M and the public key pk, samples a random element s (acting as a tag for the ciphertext) corresponding to the universe $\mathcal{U}$ of the BF and encrypts a message using pk with respect to the k positions set in the BF by s. A ciphertext is then basically identified by s and decryption works as long as at least one index pointed to by s in the BF is still set to 0. Puncturing the secret key with respect to a ciphertext (i.e., the tag s of the ciphertext) corresponds to inserting s in the BF (i.e., updating the corresponding indices to 1 and deleting the corresponding parts of the secret key). This basically means updating sk such that it no longer can decrypt any position indexed by s.

2.1 Formal Definition of Bloom Filters

A Bloom filter (BF) [7] is a probabilistic data structure for the approximate set membership problem. It allows a succinct representation T of a set $\mathcal{S}$ of elements from a large universe $\mathcal{U}$. For elements $s \in \mathcal{S}$ a query to the BF always answers 1 ("yes"). Ideally, a BF would always return 0 ("no") for elements $s \notin \mathcal{S}$, but the succinctness of the BF comes at the cost that for any query to $s \notin \mathcal{S}$ the answer can be 1, too, but only with small probability (called the *false-positive probability*).

We will only be interested in the original construction of Bloom filters by Bloom [7], and omit a general abstract definition. Instead we describe the construction from [7] directly. For a general definition refer to [26].

Definition 1 (Bloom Filter). *A* Bloom filter B *for set* $\mathcal{U}$ *consists of algorithms* $\mathsf{B} = (\mathsf{BFGen}, \mathsf{BFUpdate}, \mathsf{BFCheck})$*, which are defined as follows.*

$\mathsf{BFGen}(m, k)$**:** *This algorithm takes as input two integers* $m, k \in \mathbb{N}$*. It first samples* k *universal hash functions* $H_1, \dots, H_k$*, where* $H_j : \mathcal{U} \to [m]$*, defines* $H := (H_j)_{j\in[k]}$ *and* $T := 0^m$ *(that is,* T *is an* m*-bit array with all bits set to* 0*), and outputs* (H, T)*.*

$\mathsf{BFUpdate}(H, T, u)$**:** *Given* $H = (H_j)_{j\in[k]}$*,* $T \in \{0,1\}^m$*, and* $u \in \mathcal{U}$*, this algorithm defines the updated state* T' *by first assigning* $T' := T$*. Then, writing* $T'[i]$ *to denote the* i*-th bit of* T'*, it sets* $T'[H_j(u)] := 1$ *for all* $j \in [k]$*, and finally returns* T'*.*

$\mathsf{BFCheck}(H, T, u)$**:** *Given* $H = (H_j)_{j\in[k]}$*,* $T \in \{0,1\}^m$ *where we write* $T[i]$ *to denote the* i*-th bit of* T*, and* $u \in \mathcal{U}$*, this algorithm returns a bit* $b := \bigwedge_{j\in[k]} T[H_j(u)]$*.*

Relevant properties of Bloom filters. Let us summarize the properties of Bloom filters relevant to our work.

Perfect completeness. A Bloom filter always "recognizes" elements that have been added with probability 1. More precisely, let $\mathcal{S} = (s_1, \dots, s_n) \in \mathcal{U}^n$ be any vector of n elements of $\mathcal{U}$. Let $(H, T_0) \xleftarrow{\$} \mathsf{BFGen}(m, k)$ and define

$$T_i = \mathsf{BFUpdate}(H, T_{i-1}, s_i) \text{ for } i \in [n].$$

Then for all $s^* \in \mathcal{S}$ and all $(H, T_0) \xleftarrow{\$} \mathsf{BFGen}(m, k)$ with $m, k \in \mathbb{N}$, it holds that

$$\Pr\left[\mathsf{BFCheck}(H, T_n, s^*) = 1\right] = 1.$$

Compact representation of $\mathcal{S}$. Independent of the size of the set $\mathcal{S} \subset \mathcal{U}$ and the representation of individual elements of $\mathcal{U}$, the size of representation T is a constant number of m bits. A larger size of $\mathcal{S}$ increases only the false-positive probability, as discussed below, but not the size of the representation.

Bounded false-positive probability. The probability that an element which has not yet been added to the Bloom filter is erroneously "recognized" as being contained in the filter can be made arbitrarily small, by choosing m and k adequately, given (an upper bound on) the size of $\mathcal{S}$.
More precisely, let $\mathcal{S} = (s_1, \ldots, s_n) \in \mathcal{U}^n$ be any vector of n elements of $\mathcal{U}$. Then for any $s^* \in \mathcal{U} \setminus \mathcal{S}$, we have

$$\Pr\left[\mathsf{BFCheck}(H, T_n, s^*) = 1\right] \approx (1 - e^{-kn/m})^k,$$

where $(H, T_0) \xleftarrow{\$} \mathsf{BFGen}(m, k)$, $T_i = \mathsf{BFUpdate}(H, T_{i-1}, s_i)$ for $i \in [n]$, and the probability is taken over the random coins of BFGen.

Discussion on the choice of parameters. In order to provide a first intuition on the choice of parameters n, m and k for the use of BFs within BFE, we subsequently discuss some reasonable choices. Let us assume that we want to have $n = 2^{20}$, which amounts to adding for a full year every day about 2^{12} elements to the BF. Then, assuming the optimal number of hash functions k, and tolerating a false-positive probability of $p = 10^{-3}$, we obtain a size of the BF given by $m = {-n \ln p}/{(\ln 2)^2}$, as $m \approx 15$ Mb ≈ 2 MB. The optimal number of hash functions k is given by $k = {m}/{n} \ln 2$, and we will instantiate Bloom filters with

$$k := \lceil {m}/{n} \ln 2 \rceil .$$

This yields a correctness error $p \approx (1 - e^{-kn/m})^k = (1 - e^{-n/m \cdot \lceil \frac{m}{n} \rceil \ln 2})^k \leq 2^{-k}$. For above parameters n, m and p we obtain $k = 10$.

Looking ahead to the BFE construction in Section 2.5, at a 120-bit security level (using the pairing-friendly `BLS12-381` curve), this choice of parameters would yield ciphertexts of size < 720 B and public as well as secret keys of size < 100 B and ≈ 700 MB respectively. Thereby, we need to emphasize that initially the secret key (representing the empty BF) has its maximum size, but every puncturing (i.e., addition of an element to the BF), reduces the size of the secret key. Moreover, we stress that the false-positive probability represents an upper bound as it assumes that all $n = 2^{20}$ elements are already added to the BF, i.e., the secret key has already been punctured with respect to 2^{20} ciphertexts. Finally, when we use our time-based BFE approach (TB-BFE) from Section 2.7, we can even reduce the secret key size by reducing the maximum number of puncturings at the cost of switching the time intervals more frequently.

2.2 Formal Model of BFE

Subsequently, we introduce the formal model for BFE which essentially is a variant of puncturable encryption (PE) [19,13,11,20] with the only difference that with BFE we tolerate a non-negligible correctness error. Thus, although we are speaking of BFE, we choose to introduce a formal model for PE with a relaxed correctness definition[5] and treat BFE as an instantiation of PE. Consequently, our Definition 2 below is a variant of the one in [20], with the only difference that we allow the key generation to take the additional parameters m and k (of the BF) as input, which specify the correctness error.

For 0-RTT key establishment, our prime application in this paper, we do not need a full-blown encryption scheme, but only a key-encapsulation mechanisms (KEM) to transport a symmetric encryption key. Consequently, we chose to present our definitions by means of a puncturable KEM (PKEM). We stress that defining PKEM instead of PE does not represent any limitation, as any KEM can generically be converted into a secure full-blown encryption scheme [16]. Conversely, any secure encryption scheme trivially yields a secure KEM. Nontheless, for completeness, we give stand-alone definitions of PE tolerating a non-negligible correctness error in the full version.

Definition 2 (PKEM). *A puncturable key encapsulation (PKEM) scheme with key space $\mathcal{K}$ is a tuple* $(\mathsf{KGen}, \mathsf{Enc}, \mathsf{Punc}, \mathsf{Dec})$ *of* PPT *algorithms:*

$\mathsf{KGen}(1^\lambda, m, k)$: *Takes as input a security parameter λ, parameters m and k and outputs a secret and public key* $(\mathsf{sk}, \mathsf{pk})$ *(we assume that $\mathcal{K}$ is implicit in* pk*).*

$\mathsf{Enc}(\mathsf{pk})$: *Takes as input a public key* pk *and outputs a ciphertext C and a symmetric key* K*.*

$\mathsf{Punc}(\mathsf{sk}, C)$: *Takes as input a secret key* sk*, a ciphertext C and outputs an updated secret key* sk'*.*

$\mathsf{Dec}(\mathsf{sk}, C)$: *Takes as input a secret key* sk*, a ciphertext C and outputs a symmetric key* K *or $\perp$ if decapsulation fails.*

Correctness. We start by defining correctness of a PKEM scheme. Basically, here one requires that a ciphertext can always be decapsulated with unpunctured secret keys. However, we allow that if punctured secret keys are used for decapsulation then the probability that the decapsulation fails is bounded by some non-negligible function in the scheme's parameters m, k.

Definition 3 (Correctness). *For all $\lambda, m, k, \in \mathbb{N}$, any* $(\mathsf{sk}, \mathsf{pk}) \xleftarrow{\$} \mathsf{KGen}(1^\lambda, m, k)$ *and* $(C, \mathsf{K}) \xleftarrow{\$} \mathsf{Enc}(\mathsf{pk})$*, we have that* $\mathsf{Dec}(\mathsf{sk}, C) = \mathsf{K}$*. Moreover, for any (arbitrary interleaved) sequence $i = 1, \ldots, \ell$ (where ℓ is determined by m, k) of invocations of* $\mathsf{sk}' \xleftarrow{\$} \mathsf{Punc}(\mathsf{sk}, C')$ *for any $C' \neq C$ it holds that* $\Pr\left[\mathsf{Dec}(\mathsf{sk}', C) = \perp\right] \leq \mu(m, k)$*, where $\mu(\cdot)$ is some (possibly non-negligible) bound.*

[5] This moreover allows to compactly present our construction of forward-secret 0-RTT key exchange as this then essentially follows the argumentation in [20].

2.3 Additional Properties of a PKEM

In this section, we will define additional properties of a PKEM. One will be necessary for the application to 0-RTT key exchange from [20]. The others are required to construct a CCA-secure PKEM via the Fujisaki-Okamoto (FO) transformation, as described in Section 2.6. We will show below that our constructions of CPA-secure PKEMs satisfy these additional properties, and thus are suitable for our variant of the FO transformation, and to construct 0-RTT key exchange.

Extended correctness. Intuitively, we first require an extended variant of correctness which demands that (1) decapsulation yields a failure when attempting to decapsulate under a secret key previously punctured for that ciphertext. This is analogous to [20]. Second, we additionally demand that (2) decapsulating an honest ciphertext with the unpuctured key does always succeed and (3) if decryption does *not* fail, then the decapsulated value must match the key returned by the Enc algorithm, for any key sk' obtained from applying any sequence of puncturing operations to the initial secret key sk.

Definition 4 (Extended Correctness). *For all $\lambda, m, k, \ell \in \mathbb{N}$, any $(\mathsf{sk}, \mathsf{pk}) \xleftarrow{\$} \mathsf{KGen}(1^\lambda, m, k)$ and $(C, \mathsf{K}) \xleftarrow{\$} \mathsf{Enc}(\mathsf{pk})$ and any (arbitrary interleaved and possibly empty) sequence $C_1, \ldots, C_\ell$ of invocations of $\mathsf{sk}' \xleftarrow{\$} \mathsf{Punc}(\mathsf{sk}, C_i)$ it holds that:*

1. ***Impossibility of false-negatives:***
   $\mathsf{Dec}(\mathsf{sk}', C_i) = \perp$ *for all* $i \in [\ell]$.
2. ***Perfect correctness of the initial, non-punctured secret key:***
   If $(C, \mathsf{K}) \xleftarrow{\$} \mathsf{Enc}(\mathsf{pk})$ *then* $\mathsf{Dec}(\mathsf{sk}, C) = \mathsf{K}$*, where* sk *is the initial, non-punctured secret key.*
3. ***Semi-correctness of punctured secret keys:***
   If $\mathsf{Dec}(\mathsf{sk}', C) \neq \perp$ *then* $\mathsf{Dec}(\mathsf{sk}', C) = \mathsf{Dec}(\mathsf{sk}, C)$.

Separable randomness. We require that the encapsulation algorithm Enc essentially reads the key K in $(C, \mathsf{K}) \xleftarrow{\$} \mathsf{Enc}(\mathsf{pk})$ directly from its random input tape. Intuitively, this will later enable us to make the randomness r used by the encapsulation algorithm Enc dependent on the key K computed by Enc.

Definition 5 (Separable Randomness). *Let* $\mathsf{PKEM} = (\mathsf{KGen}, \mathsf{Enc}, \mathsf{Punc}, \mathsf{Dec})$ *be a PKEM. We say that* PKEM *has* separable randomness*, if one can equivalently write the encapsulation algorithm* Enc *as*

$$(C, \mathsf{K}) \xleftarrow{\$} \mathsf{Enc}(\mathsf{pk}) = \mathsf{Enc}(\mathsf{pk}; (r, \mathsf{K})),$$

for uniformly random $(r, \mathsf{K}) \in \{0,1\}^{\rho+\lambda}$*, where* $\mathsf{Enc}(\cdot\,;\cdot)$ *is a deterministic algorithm whose output is uniquely determined by* pk *and the randomness* $(r, \mathsf{K}) \in \{0,1\}^{\rho+\lambda}$.

Remark. We note that one can generically construct a separable PKEM from any non-separable PKEM. Given a non-separable PKEM with encapsulation algorithm Enc, a separable PKEM with encryption algorithm Enc' can be obtained as follows:

$\mathsf{Enc}'(\mathsf{pk}; (r, \mathsf{K}')) :$ Run $(C, \mathsf{K}) \xleftarrow{\$} \mathsf{Enc}(\mathsf{pk}; r)$, set $C' := (C, \mathsf{K} \oplus \mathsf{K}')$ return (C', K').

We need separability in order to apply our variant of the FO transformation, which is the reason why we have to make it explicit. Alternatively, we could have started from a non-separable PKEM and applied the above construction. However, this adds an additional component to the ciphertext, while the construction given in Section 2.5 will already be separable, such that we can avoid this overhead.

Publicly-checkable puncturing. Finally, we need that it is efficiently checkable whether the decapsulation algorithm outputs $\perp = \mathsf{Dec}(\mathsf{sk}, C)$, given *not* the secret key sk, but only the public key pk, the ciphertext C to be decrypted, and the sequence $C_1, \ldots, C_w$ at which the secret key sk has been punctured.

Definition 6 (Publicly-Checkable Puncturing). *Let $\mathcal{Q} = (C_1, \ldots, C_w)$ be any list of ciphertexts. We say that* PKEM *allows* publicly-checkable puncturing*, if there exists an efficient algorithm* CheckPunct *with the following correctness property.*

1. *Run* $(\mathsf{sk}, \mathsf{pk}) \xleftarrow{\$} \mathsf{KGen}(1^\lambda, m, k)$.
2. *Compute* $C_i \xleftarrow{\$} \mathsf{Enc}(\mathsf{pk})$ *and* $\mathsf{sk} = \mathsf{Punc}(\mathsf{sk}, C_i)$ *for* $i \in [w]$.
3. *Let C be any string. We require that*

$$\perp = \mathsf{Dec}(\mathsf{sk}, C) \iff \perp = \mathsf{CheckPunct}(\mathsf{pk}, \mathcal{Q}, C).$$

From a high-level perspective, this additional property will be necessary to simulate the decryption oracle properly in the CCA security experiment when our variant of the FO transformation is applied. Together with the second and third property of Definition 4, it replaces the perfect correctness property required in the original FO transformation.

Min-entropy of ciphertexts. Following [23], we require that ciphertexts of a randomness-separable PKEM have sufficient min-entropy, even if K is fixed:

Definition 7 (γ-Spreadness). *Let* $\mathsf{PKEM} = (\mathsf{KGen}, \mathsf{Enc}, \mathsf{Punc}, \mathsf{Dec})$ *be a* randomness-separable *PKEM with ciphertext space $\mathcal{C}$. We say that* PKEM *is* γ-spread*, if for any honestly generated* pk*, any key* K *and any* $C \in \mathcal{C}$

$$\Pr_{r \xleftarrow{\$} \{0,1\}^\rho} [C = \mathsf{Enc}(\mathsf{pk}; (r, \mathsf{K}))] \leq 2^{-\gamma}.$$

2.4 Security Definitions

We define three notions of security for PKEMs. The two "standard" security notions are indistinguishability under chosen-plaintext (IND-CPA) and chosen-ciphertext (IND-CCA) attacks. We also consider one-wayness under chosen-plaintext attacks (OW-CPA). The latter is the weakest notion among the ones considered in this paper, and implied by both IND-CPA and IND-CCA, but sufficient for our generic construction of IND-CCA-secure PKEMs.

Indistinguishability-based security. Figure 1 defines the IND-CPA and IND-CCA experiments for PKEMs. The experiments are similar to the security notions for conventional KEMs, but the adversary can arbitrarily puncture the secret key via the Punc oracle and retrieve the punctured secret key via the Corr oracle, once it has been punctured on the challenge ciphertext C^*.

```
$\mathbf{Exp}^{\mathsf{T}}_{\mathcal{A},\mathsf{PKEM}}(\lambda, m, k)$:
  $(\mathsf{sk}, \mathsf{pk}) \xleftarrow{\$} \mathsf{KGen}(1^\lambda, m, k)$, $(C^*, \mathsf{K}_0) \xleftarrow{\$} \mathsf{Enc}(\mathsf{pk})$, $\mathcal{Q} \leftarrow \emptyset$
  $\mathsf{K}_1 \xleftarrow{\$} \mathcal{K}$, $b \xleftarrow{\$} \{0,1\}$
  $b^* \xleftarrow{\$} \mathcal{A}^{\mathcal{O},\mathsf{Punc}(\mathsf{sk},\cdot),\mathsf{Corr}}(\mathsf{pk}, C^*, \mathsf{K}_b)$
    where $\mathcal{O} \leftarrow \{\mathsf{Dec}'(\mathsf{sk},\cdot)\}$ if $\mathsf{T} = \mathsf{IND\text{-}CCA}$ and $\mathcal{O} \leftarrow \emptyset$ otherwise.
    $\mathsf{Dec}'(\mathsf{sk}, C)$ behaves as $\mathsf{Dec}$ but returns $\perp$ if $C = C^*$
    $\mathsf{Punc}(\mathsf{sk}, C)$ runs $\mathsf{sk} \xleftarrow{\$} \mathsf{Punc}(\mathsf{sk}, C)$ and $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{C\}$
    $\mathsf{Corr}$ returns $\mathsf{sk}$ if $C^* \in \mathcal{Q}$ and $\perp$ otherwise
  If $b^* = b$ then return 1
  return 0
```

Fig. 1. Indistinguishability-based security for PKEMs.

Definition 8 (Indistinguishability-Based Security of PKEM). *For* $\mathsf{T} \in \{\mathsf{IND\text{-}CPA}, \mathsf{IND\text{-}CCA}\}$*, we define the advantage of an adversary* $\mathcal{A}$ *in the* T *experiment* $\mathbf{Exp}^{\mathsf{T}}_{\mathcal{A},\mathsf{PKEM}}(\lambda, m, k)$ *as*

$$\mathbf{Adv}^{\mathsf{T}}_{\mathcal{A},\mathsf{PKEM}}(\lambda, m, k) := \left| \Pr\left[\mathbf{Exp}^{\mathsf{T}}_{\mathcal{A},\mathsf{PKEM}}(\lambda, m, k) = 1\right] - \frac{1}{2} \right|.$$

A puncturable key-encapsulation scheme PKEM is $\mathsf{T} \in \{\mathsf{IND\text{-}CPA}, \mathsf{IND\text{-}CCA}\}$ secure, if $\mathbf{Adv}^{\mathsf{T}}_{\mathcal{A},\mathsf{PKEM}}(\lambda, m, k)$ is a negligible function in λ for all $m, k > 0$ and all PPT adversaries $\mathcal{A}$.

One-wayness under chosen-plaintext attack. Figure 2 defines the OW-CPA experiment. The experiment is similar to the IND-CPA experiment, except that the goal of the adversary is to recover the encapsulated key, given a random challenge ciphertext.

```
$\mathbf{Exp}^{\mathsf{OW\text{-}CPA}}_{\mathcal{A},\mathsf{PKEM}}(\lambda, m, k)$:
  $(\mathsf{sk}, \mathsf{pk}) \xleftarrow{\$} \mathsf{KGen}(1^\lambda, m, k)$, $(C^*, \mathsf{K}_0) \xleftarrow{\$} \mathsf{Enc}(\mathsf{pk})$, $\mathcal{Q} \leftarrow \emptyset$
  $\mathsf{K}_0^* \xleftarrow{\$} \mathcal{A}^{\mathsf{Punc}(\mathsf{sk},\cdot),\mathsf{Corr}}(\mathsf{pk}, C^*)$
    where $\mathsf{Punc}(\mathsf{sk}, C)$ runs $\mathsf{sk} \xleftarrow{\$} \mathsf{Punc}(\mathsf{sk}, C)$ and $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{C\}$
    $\mathsf{Corr}$ returns $\mathsf{sk}$ if $C^* \in \mathcal{Q}$ and $\perp$ otherwise
  If $\mathsf{K}_0^* = \mathsf{K}_0$ then return 1
  return 0
```

Fig. 2. OW-CPA security for PKEMs.

Definition 9 (One-Wayness Under Chosen-Plaintext Attack). *We define the advantage of an adversary $\mathcal{A}$ in experiment* $\mathbf{Exp}^{\mathsf{OW\text{-}CPA}}_{\mathcal{A},\mathsf{PKEM}}(\lambda, m, k)$ *as*

$$\mathbf{Adv}^{\mathsf{OW\text{-}CPA}}_{\mathcal{A},\mathsf{PKEM}}(\lambda, m, k) := \Pr\left[\mathbf{Exp}^{\mathsf{OW\text{-}CPA}}_{\mathcal{A},\mathsf{PKEM}}(\lambda, m, k) = 1\right].$$

A PKEM is $\mathsf{OW\text{-}CPA}$ secure, if $\mathbf{Adv}^{\mathsf{OW\text{-}CPA}}_{\mathcal{A},\mathsf{PKEM}}(\lambda, m, k)$ is a negligible function in λ for all $m, k > 0$ and all PPT adversaries $\mathcal{A}$.

2.5 Basic Bloom Filter Encryption

Bilinear maps and notation. In the sequel, let BilGen be an algorithm that, on input a security parameter 1^λ, outputs $(p, e, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2) \xleftarrow{\$} \mathsf{BilGen}(1^\lambda)$, where $\mathbb{G}_1$, $\mathbb{G}_2$, $\mathbb{G}_T$ are groups of prime order p with bilinear map $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ and generators $g_i \in \mathbb{G}_i$ for $i \in \{1, 2\}$.

Construction. In the sequel, let $\mathsf{Params} := (p, e, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2) \xleftarrow{\$} \mathsf{BilGen}(1^\lambda)$, and $g_T = e(g_1, g_2)$. We will always assume that all algorithms described below implicitly receive these parameters as additional input. Let $\mathsf{B} = (\mathsf{BFGen}, \mathsf{BFUpdate}, \mathsf{BFCheck})$ be a Bloom filter for set $\mathbb{G}_1$. Furthermore, let $G : \mathbb{N} \to \mathbb{G}_2$ and $G' : \mathbb{G}_T \to \{0, 1\}^\lambda$ be cryptographic hash functions (which will be modelled as random oracles [5] in the security proof).

Let $\mathsf{PKEM} = (\mathsf{KGen}, \mathsf{Enc}, \mathsf{Punc}, \mathsf{Dec})$ be defined as follows.

$\underline{\mathsf{KGen}(1^\lambda, m, k)}$: This algorithm first generates a Bloom filter instance by running $(H, T) \xleftarrow{\$} \mathsf{BFGen}(m, k)$. Then it chooses $\alpha \xleftarrow{\$} \mathbb{Z}_p$, and computes and returns

$$\mathsf{sk} := (T, (G(i)^\alpha)_{i \in [m]}) \text{ and } \mathsf{pk} := (g_1^\alpha, H).$$

Remark. The reader familiar with the Boneh-Franklin IBE scheme [9] may note that the secret key contains m elements of $\mathbb{G}_2$, each essentially being a secret key of the Boneh-Franklin scheme for "identity" i, $i \in [m]$, with respect to "master public-key" g_1^α.

$\underline{\mathsf{Enc}(\mathsf{pk})}$: This algorithm takes as input a public key pk of the above form. It samples a uniformly random key $\mathsf{K} \xleftarrow{\$} \{0, 1\}^\lambda$ and exponent $r \xleftarrow{\$} \mathbb{Z}_p$. Then it computes $i_j := H_j(g_1^r)$ for $(H_j)_{j \in [k]} := H$, then $y_j = e(g_1^\alpha, G(i_j))^r$ for $j \in [k]$, and finally

$$C := \left(g_1^r, (G'(y_j) \oplus \mathsf{K})_{j \in [k]}\right).$$

It outputs $(C, \mathsf{K}) \in (\mathbb{G}_1 \times \{0, 1\}^{k\lambda}) \times \{0, 1\}^\lambda$.

Remark. Note that for each $j \in [k]$, the tuple $(g_1^r, G'(y_j) \oplus \mathsf{K})$ is essentially a "hashed Boneh-Franklin IBE" ciphertext, encrypting K for "identity" $i_j = H_j(g_1^r)$ and with respect to master public key g_1^α, where the identity is derived deterministically from a "unique" (with overwhelming probability) ciphertext component g_1^r. Thus, the ciphertext C essentially consists of k Boneh-Franklin ciphertexts that share the same randomness r, each encrypting the same key K for an "identity" derived deterministically from g_1^r.

Note also that this construction of Enc satisfies the requirement of separable randomness from Definition 5. Furthermore, ciphertexts are γ-spread according to Definition 7 with $\gamma = \log_2 p$, because g_1^r is uniformly distributed over $\mathbb{G}_1$.

$\underline{\mathsf{Punc}(\mathsf{sk}, C)}$: Given a ciphertext $C := \left(g_1^r, (G'(y_j) \oplus \mathsf{K})_{j \in [k]}\right)$ and secret key $\mathsf{sk} = (T, (\mathsf{sk}[i])_{i \in [m]})$, the puncturing algorithm first computes $T' = \mathsf{BFUpdate}(H, T, g_1^r)$. Then, for each $i \in [m]$ it defines

$$\mathsf{sk}'[i] := \begin{cases} \mathsf{sk}[i] & \text{if } T'[i] = 0, \text{ and} \\ \perp & \text{if } T'[i] = 1, \end{cases}$$

where $T'[i]$ denotes the i-th bit of T'. Finally, this algorithm returns

$$\mathsf{sk}' := (T', (\mathsf{sk}'[i])_{i \in [m]}).$$

Remark. Note that the above procedure is correct even if the procedure is applied repeatedly with different ciphertexts C, since the $\mathsf{BFUpdate}$ algorithm only changes bits of T from 0 to 1, but never from 1 to 0. So we can delete a secret key element $\mathsf{sk}[i]$ once $T'[i]$ has been set to 1. Furthermore, we have $\mathsf{sk}'[i] = \perp \iff T'[i] = 1$. Intuitively, this will ensure that we can use this key to decrypt a ciphertext $C := \left(g_1^r, (G'(y_j) \oplus \mathsf{K})_{j \in [k]}\right)$ if and only if $\mathsf{BFCheck}(H, T, g_1^r) = 0$, where (H, T) is the Bloom filter instance contained in the public key. Note also that the puncturing algorithm essentially only evaluates k universal hash functions $H = (H_j)_{j \in [k]}$ and then deletes a few secret keys, which makes this procedure extremely efficient. Finally, observe that the filter state T can be efficiently re-computed given only public information, namely the list of hash functions H contained in pk and the sequence of ciphertexts $C_1, \ldots, C_w$ on which a secret key has been punctured. This yields the existence of an efficient $\mathsf{CheckPunct}$ according to Definition 6.

$\underline{\mathsf{Dec}(\mathsf{sk}, C)}$: Given a secret key $\mathsf{sk} = (T, (\mathsf{sk}[i])_{i \in [m]})$ and a ciphertext $C := (C[0], C[i_1], \ldots, C[i_k])$ it first checks whether $\mathsf{BFCheck}(H, T, C[0]) = 1$, and outputs $\perp$ in this case. Otherwise, note that $\mathsf{BFCheck}(H, T, C[0]) = 0$ implies that there exists at least one index i^* with $\mathsf{sk}[i^*] \neq \perp$. It picks the smallest index $i^* \in \{i_1, \ldots, i_k\}$ such that $\mathsf{sk}[i^*] = G(i^*)^\alpha \neq \perp$, computes

$$y_{i^*} := e(g_1^r, G(i^*)^\alpha),$$

and returns $\mathsf{K} := C[i^*] \oplus G'(y_{i^*})$.

Remark. If $\mathsf{BFCheck}(H, T_n, C[0]) = 0$, then the decryption algorithm performs a "hashed Boneh-Franklin" decryption with a secret key for one of the identities. Note that $\mathsf{Dec}(\mathsf{sk}_n, C) \neq \perp \iff \mathsf{BFCheck}(H, T, C[0]) = 0$, which guarantees the first extended correctness property required by Definition 4. It is straightforward to verify that the other two extended correctness properties of Definition 4 hold as well.

Design choices. We note that we have chosen to base our Bloom filter encryption scheme on *hashed* Boneh-Franklin IBE instead of standard Boneh-Franklin for two reasons. First, it allows us to keep ciphertexts short and independent of the size of the binary representation of elements of $\mathbb{G}_T$. This is useful, because the recent advances for computing discrete logarithms in finite extension fields [24] apply to the target group of state-of-the-art pairing-friendly elliptic curve groups. Recent assessments of the impact of these advances by Menezes et al. [25] as well as Barbulescu and Duquesne [2] suggest that for currently used efficient curve families such as BN [4] or BLS [3] curves a conservative choice of parameters for the 128 bit security level yields sizes of $\mathbb{G}_T$ elements of $\approx 4600 - 5500$ bits. The hash function allows us to "compress" these group elements in the ciphertext to 128 bits. Even if future research enables the construction of bilinear maps where elements of $\mathbb{G}_T$ can be represented by 2λ bits for λ-bit security (which is optimal), it is still preferable to hash group elements to λ bits to reduce the ciphertext by a factor of about 2. Second, by modelling G' as a random oracle, we can reduce security to a weaker complexity assumption.

Correctness error of this scheme. We will now explain that the correctness error of this scheme is essentially identical to the false-positive probability of the Bloom filter, up to a statistically small distance which corresponds to the probability that two independent ciphertexts share the same randomness r.

For $m, k \in \mathbb{N}$, let $(\mathsf{sk}_0, \mathsf{pk}) \xleftarrow{\$} \mathsf{KGen}(1^\lambda, m, k)$, let $\mathcal{U} := \{C : (C, \mathsf{K}) \xleftarrow{\$} \mathsf{Enc}(\mathsf{pk})\}$ denote the set of all valid ciphertext with respect to pk. Let $\mathcal{S} = (C_1, \ldots, C_n)$ be a list of n ciphertexts, where $(C_i, \mathsf{K}_i) \xleftarrow{\$} \mathsf{Enc}(\mathsf{pk})$, and run $\mathsf{sk}_i = \mathsf{Punc}(\mathsf{sk}_{i-1}, C_i)$ for $i \in [n]$ to determine the secret key sk_n obtained from puncturing sk_0 iteratively on all ciphertexts $C_i \in \mathcal{S}$.

Now let us consider the probability

$$\Pr\left[\mathsf{Dec}(\mathsf{sk}_n, C^*) \neq \mathsf{K}^* : (C^*, \mathsf{K}^*) \xleftarrow{\$} \mathsf{Enc}(\mathsf{pk}), C^* \notin \mathcal{S}\right]$$

that a newly generated ciphertext $C^* \notin \mathcal{S}$ is not correctly decrypted by sk_n. To this end, let $C^*[0] = g_1^{r^*}$ denote the first component of ciphertext $C^* = (g_1^{r^*}, C_1^*, \ldots, C_k^*)$, and likewise let $C_i[0]$ denote the first component of ciphertext C_i for all $C_i \in \mathcal{S}$. Writing $\mathsf{sk}_n = (T_n, (\mathsf{sk}_n[i])_{i \in [m]})$ and $\mathsf{pk} = (g_1^\alpha, H)$, one can now verify that we have $\mathsf{Dec}(\mathsf{sk}_n, C^*) \neq \mathsf{K}^* \iff \mathsf{BFCheck}(H, T_n, C^*[0]) = 1$, because $\mathsf{BFCheck}(H, T_n, C^*[0]) = 0$ guarantees that there exists at least one index j such that $\mathsf{sk}_n[H_j(C^*[0])] \neq \bot$, so correctness of decryption follows essentially from correctness of the Boneh-Franklin scheme. Thus, we have to consider the probability that $\mathsf{BFCheck}(H, T_n, C^*[0]) = 1$. We distinguish between two cases:

1. There exists an index $i \in [n]$ such that $C^*[0] = C_i[0]$. Note that this implies immediately that $\mathsf{BFCheck}(H, T_n, C^*[0]) = 1$. However, recall that $C^*[0] = g_1^{r^*}$ is a uniformly random element of $\mathbb{G}_1$. Therefore the probability that this happens is upper bounded by n/p, which is negligibly small.
2. $C^*[0] \neq C_i[0]$ for all $i \in [n]$. In this case, as explained in Section 2.1, the soundness of the Bloom filter guarantees that $\Pr[\mathsf{BFCheck}(H, T_n, C^*[0]) = 1] \approx 2^{-k}$.

In summary, the correctness error of this scheme is approximately $2^{-k} + n/p$. Since n/p is negligibly small, this essentially amounts to the correctness error of the Bloom filter, which in turn depends on the number of ciphertexts n, and the choice of parameters m, k.

Flexible instantiability of this scheme. Our scheme is highly parameterizable in the sense that we can adjust the size of keys and ciphertexts by adjusting the correctness error (determined by the choice of parameters m, k that in turn determine the false-positive probability of the Bloom filter) of our scheme.

Additional properties. As already explained in the remarks after the description of the individual algorithms of PKEM, the scheme satisfies the requirements of Definitions 4, 5, 6, and 7.

$\mathsf{IND\text{-}CPA}$-security. We base $\mathsf{IND\text{-}CPA}$-security on a bilinear computational Diffie-Hellman variant in the bilinear groups generated by BilGen.

Definition 10 (BCDH). *We define the advantage of adversary $\mathcal{A}$ in solving the BCDH problem with respect to* BilGen *as*

$$\mathbf{Adv}^{\mathsf{BCDH}}_{\mathcal{A},\mathsf{BilGen}}(\lambda) := \Pr\left[e(g_1, h_2)^{r\alpha} \xleftarrow{\$} \mathcal{A}(\mathsf{Params}, g_1^r, g_1^\alpha, g_2^\alpha, h_2)\right],$$

where $\mathsf{Params} = (p, e, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2) \xleftarrow{\$} \mathsf{BilGen}(1^\lambda)$, and $(g_1^r, g_1^\alpha, g_2^\alpha, h_2) \xleftarrow{\$} \mathbb{G}_1^2 \times \mathbb{G}_2$.

Theorem 1. *From each efficient adversary $\mathcal{B}$ that issues q queries to random oracle G' we can construct an efficient adversary $\mathcal{A}$ with*

$$\mathbf{Adv}^{\mathsf{BCDH}}_{\mathcal{A},\mathsf{BilGen}}(\lambda) \geq \frac{\mathbf{Adv}^{\mathsf{IND\text{-}CPA}}_{\mathcal{B},\mathsf{PKEM}}(\lambda, m, k)}{kq}.$$

Proof. Algorithm $\mathcal{A}$ receives as input a BCDH-challenge tuple $(g_1^r, g_1^\alpha, g_2^\alpha, h_2)$. It runs adversary $\mathcal{B}$ as a subroutine by simulating the $\mathbf{Exp}^{\mathsf{IND\text{-}CPA}}_{\mathcal{B},\mathsf{PKEM}}(\lambda, m, k)$ experiment, including random oracles G and G', as follows.

First, it defines $\mathcal{Q} := \emptyset$, runs $(H, T) \xleftarrow{\$} \mathsf{BFGen}(m, k)$, and defines the public key as $\mathsf{pk} := (g_1^\alpha, H)$. Note that this public key is identically distributed to a public key output by $\mathsf{KGen}(1^\lambda, m, k)$. In order to simulate the challenge ciphertext, the adversary chooses a random key $\mathsf{K} \xleftarrow{\$} \{0,1\}^\lambda$ and k uniformly random values $Y_j \xleftarrow{\$} \{0,1\}^\lambda$, $j \in [k]$, and defines the challenge ciphertext as $C^* := (g_1^r, (Y_j)_{j \in [k]})$. Finally, it outputs $(\mathsf{pk}, C^*, \mathsf{K})$ to $\mathcal{B}$.

Whenever $\mathcal{B}$ queries $\mathsf{Punc}(\mathsf{sk}, \cdot)$ on input $C = (C[0], \ldots)$, then $\mathcal{A}$ updates T by running $T = \mathsf{BFUpdate}(H, T, C[0])$, and $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{C\}$.

Whenever a random oracle query to $G : \mathbb{N} \to \mathbb{G}_2$ is made (either by $\mathcal{A}$ or $\mathcal{B}$), with input $\ell \in \mathbb{N}$, then $\mathcal{A}$ responds with $G(\ell)$, if $G(\ell)$ has already been defined. If not, then $\mathcal{A}$ chooses a random integer $r_\ell \xleftarrow{\$} \mathbb{Z}_p$, and returns $G(\ell)$, where

$$G(\ell) := \begin{cases} h_2 \cdot g_2^{r_\ell} & \text{if } \ell \in \{H_j(g_1^r) : j \in [k]\}, \text{ and} \\ g_2^{r_\ell} & \text{otherwise.} \end{cases}$$

This definition of G allows $\mathcal{A}$ to simulate the Corr oracle as follows. When $\mathcal{B}$ queries Corr, then it first checks whether $C^* \in \mathcal{Q}$, and returns $\perp$ if this does not hold. Otherwise, note that we must have $\forall j \in [k] : T[H_j(g_1^r)] = 0$, where $H = (H_j)_{j\in[k]}$ and $T[\ell]$ denotes the ℓ-th bit of T. Thus, by the simulation of G described above, $\mathcal{A}$ is able to compute and return $G(\ell)^\alpha = (g_2^{r_\ell})^\alpha = (g_2^\alpha)^{r_\ell}$ for all ℓ with $\ell \notin \{H_j(g_1^r) : j \in [k]\}$, and therefore in particular for all ℓ with $T[\ell] = 1$. This enables the perfect simulation of Corr.

Finally, whenever $\mathcal{B}$ queries random oracle $G' : \mathbb{G}_T \to \{0,1\}^\lambda$ on input y, then $\mathcal{A}$ responds with $G'(y)$, if $G'(y)$ has already been defined. If not, then $\mathcal{A}$ chooses a random string $Y \xleftarrow{\$} \{0,1\}^\lambda$, assigns $G'(y) := Y$, and returns $G'(y)$. Now we have to distinguish between two types of adversaries.

1. A Type-1 adversary $\mathcal{B}$ never queries G' on input of a value y, such that there exists $j \in [k]$ such that $y = e(g_1^\alpha, G(H_j(g_1^r)))^r$. Note that in this case the value $Y_j' := G'(e(g_1^\alpha, G(H_j(g_1^r))))$ remains undefined for all $j \in [k]$ throughout the entire experiment. Thus, information-theoretically, a Type-1 adversary receives no information about the key encrypted in the challenge ciphertext C^*, and thus can only have advantage $\mathbf{Adv}_{\mathcal{B},\mathsf{PKEM}}^{\mathsf{IND\text{-}CPA}}(\lambda, m, k) = 0$, in which case the theorem holds trivially.
2. A Type-2 adversary queries $G'(y)$ such that there exists $j \in [k]$ with $y = e(g_1^\alpha, G(H_j(g_1^r)))^r$. $\mathcal{A}$ uses a Type-2 adversary to solve the BCDH challenge as follows. At the beginning of the game, it picks two indices $(q^*, j^*) \xleftarrow{\$} [q] \times [k]$ uniformly random. When $\mathcal{B}$ outputs y in its q^*-th query to G', then $\mathcal{A}$ computes and outputs $W := y \cdot e(g_1^\alpha, g_2^r)^{-r_\ell}$. Since $\mathcal{B}$ is a Type-2 adversary, we know that at some point it will query $G'(y)$ with $y = e(g_1^\alpha, G(H_j(g_1^r)))^r$ for some $j \in [k]$. If this is the q^*-th query and we have $j = j^*$, which happens with probability $1/(qk)$, then we have
$$\begin{aligned} W &= y \cdot e(g_1^\alpha, g_2^r)^{-r_\ell} = e(g_1^\alpha, G(H_j(g_1^r)))^r \cdot e(g_1^\alpha, g_2^r)^{-r_\ell} \\ &= e(g_1^\alpha, h_2 \cdot g_2^{r_\ell})^r \cdot e(g_1^\alpha, g_2^r)^{-r_\ell} = e(g_1^\alpha, h_2)^r \cdot e(g_1^\alpha, g_2^{r_\ell})^r \cdot e(g_1^\alpha, g_2^r)^{-r_\ell} \end{aligned}$$
and thus W is a solution to the given BCDH instance. □

$\mathsf{OW\text{-}CPA}$-Security. The following theorem can either be proven analogous to Theorem 1, or based on the fact that $\mathsf{IND\text{-}CPA}$-security implies $\mathsf{OW\text{-}CPA}$-security. Therefore we give it without proof.

Theorem 2. *From each efficient adversary $\mathcal{B}$ that issues q queries to random oracle G' we can construct an efficient adversary $\mathcal{A}$ with*
$$\mathbf{Adv}_{\mathcal{A},\mathsf{BilGen}}^{\mathsf{BCDH}}(\lambda) \geq \frac{\mathbf{Adv}_{\mathcal{B},\mathsf{PKEM}}^{\mathsf{OW\text{-}CPA}}(\lambda, m, k)}{kq}.$$

Remark 1. The construction presented above allows to switch the roles of $\mathbb{G}_1$ and $\mathbb{G}_2$, i.e., to switch all elements in $\mathbb{G}_1$ to $\mathbb{G}_2$ and vice versa. This might be beneficial regarding the size of the secret key when instantiating our construction using a bilinear group where the representation of elements in $\mathbb{G}_2$ requires more space than the representation of elements in $\mathbb{G}_1$.

2.6 CCA-Security via Fujisaki-Okamoto

We obtain a CCA-secure PKEM by adopting the Fujisaki-Okamoto (FO) transformation [16] to the PKEM setting. Since the FO transformation does not work generically for any KEM, we have to use the additional requirements on the underlying PKEM that have been defined in Section 2.3. These additional properties enable us to overcome the difficulty that the original Fujisaki-Okamoto transformation from [16] requires *perfect* correctness, what no puncturable KEM can provide. We note that Hofheinz *et al.* [23] give a new, modular analysis of the FO transformation, which also works for public key *encryption* schemes with *negligible* correctness error, however, it is not applicable to PKEMs with non-negligible correctness error because the bounds given in [23] provide insufficient security in this case.

Construction. Let $\mathsf{PKEM} = (\mathsf{KGen}, \mathsf{Enc}, \mathsf{Punc}, \mathsf{Dec})$ be a PKEM with *separable randomness* according to Definition 5. Recall that this means that we can write Enc equivalently as $(C, \mathsf{K}) \xleftarrow{\$} \mathsf{Enc}(\mathsf{pk}) = \mathsf{Enc}(\mathsf{pk}; (r, \mathsf{K}))$ for uniformly random $(r, \mathsf{K}) \xleftarrow{\$} \{0,1\}^{\rho+\lambda}$. In the sequel, let R be a hash function (modeled as a random oracle in the security proof), mapping $R : \{0,1\}^* \to \{0,1\}^{\rho+\lambda}$. We construct a new scheme $\mathsf{PKEM}' = (\mathsf{KGen}', \mathsf{Enc}', \mathsf{Punc}', \mathsf{Dec}')$ as follows.

$\underline{\mathsf{KGen}'(1^\lambda, m, k)}$: This algorithm is identical to KGen.

$\underline{\mathsf{Enc}'(\mathsf{pk})}$: Algorithm Enc' samples $\mathsf{K} \xleftarrow{\$} \{0,1\}^\lambda$. Then it computes $(r, \mathsf{K}') := R(\mathsf{K}) \in \{0,1\}^{\rho+\lambda}$, runs $(C, \mathsf{K}) \xleftarrow{\$} \mathsf{Enc}(\mathsf{pk}; (r, \mathsf{K}))$, and returns (C, K').

$\underline{\mathsf{Punc}'(\mathsf{sk}, C)}$: This algorithm is identical to Punc.

$\underline{\mathsf{Dec}'(\mathsf{sk}, C)}$: This algorithm first runs $\mathsf{K} \xleftarrow{\$} \mathsf{Dec}(\mathsf{sk}, C)$, and returns $\perp$ if $\mathsf{K} = \perp$. Otherwise, it computes $(r, \mathsf{K}') = R(\mathsf{K})$, and checks consistency of the ciphertext by verifying that $(C, \mathsf{K}) = \mathsf{Enc}(\mathsf{pk}; (r, \mathsf{K}))$. If this does not hold, then it outputs $\perp$. Otherwise it outputs K'.

Correctness error and extended correctness. Both the correctness error and the extended correctness according to Definition 4 are not affected by the Fujisaki-Okamoto transform. Therefore these properties are inherited from the underlying scheme. The fact that the first property of Definition 4 is satisfied makes the scheme suitable for the application to 0-RTT key establishment.

IND-CCA-security. The security proof reduces security of our modified scheme to the OW-CPA-security of the scheme from Section 2.5.

Theorem 3. *Let* $\mathsf{PKEM} = (\mathsf{KGen}, \mathsf{Enc}, \mathsf{Punc}, \mathsf{Dec})$ *be a BFKEM scheme that satisfies the additional properties of Definitions 4 and 6, and which is* γ*-spread according to Definition 7. Let* $\mathsf{PKEM}' = (\mathsf{KGen}', \mathsf{Enc}', \mathsf{Punc}', \mathsf{Dec}')$ *be the scheme described in Section 2.6. From each efficient adversary* $\mathcal{A}$ *that issues at most* $q_\mathcal{O}$ *queries to oracle* $\mathcal{O}$ *and* q_R *queries to random oracle* R*, we can construct an efficient adversary* $\mathcal{B}$ *with*

$$\mathbf{Adv}^{\mathsf{OW\text{-}CPA}}_{\mathcal{B},\mathsf{PKEM}}(\lambda, m, k) \geq \frac{\mathbf{Adv}^{\mathsf{IND\text{-}CCA}}_{\mathcal{A},\mathsf{PKEM}'}(\lambda, m, k) - q_\mathcal{O}/2^\gamma}{q_R}.$$

Proof. We proceed in a sequence of games. In the sequel, $\mathcal{O}_i$ is the implementation of the decryption oracle in Game i.

Game 0. This is the original $\mathsf{IND\text{-}CCA}$ security experiment from Definition 8, played with the scheme described above. In particular, the decryption oracle $\mathcal{O}_0$ is implemented as follows:

```
$\mathcal{O}_0(C)$

$\mathsf{K} \xleftarrow{\$} \mathsf{Dec}(\mathsf{sk}, C)$
If $\mathsf{K} = \perp$ then return $\perp$
$(r, \mathsf{K}') = R(\mathsf{K})$
If $(C, \mathsf{K}) \neq \mathsf{Enc}(\mathsf{pk}; (r, \mathsf{K}))$ then return $\perp$
Return $\mathsf{K}'$
```

Recall that K_0 denotes the encapsulated key computed by the $\mathsf{IND\text{-}CCA}$ experiment. K_0 is uniquely defined by the challenge ciphertext C^* via $\mathsf{K}_0 := \mathsf{Dec}(\mathsf{sk}_0, C^*)$, where sk_0 is the initial (non-punctured) secret key, since the scheme satisfies extended correctness (Definition 4, second property). Let Q_0 denote the event that $\mathcal{A}$ ever queries K_0 to random oracle R. Note that $\mathcal{A}$ has zero advantage in distinguishing K' from random, until Q_0 occurs, because R is a random function. Thus, we have $\Pr[Q_0] \geq \mathbf{Adv}^{\mathsf{IND\text{-}CCA}}_{\mathcal{A},\mathsf{PKEM}'}(\lambda, m, k)$. In the sequel, we denote with Q_i the event that $\mathcal{A}$ ever queries K_0 to random oracle R in Game i.

Game 1. This game is identical to Game 0, except that after computing $\mathsf{K} \xleftarrow{\$} \mathsf{Dec}(\mathsf{sk}, C)$ and checking whether $\mathsf{K} \neq \perp$, the experiment additionally checks whether the adversary has ever queried random oracle R on input K, and returns $\perp$ if not. More precisely, the experiment maintains a list

$$L_R = \{(\mathsf{K}, (r, \mathsf{K}')) : \mathcal{A} \text{ queried } R(\mathsf{K}) = (r, \mathsf{K}')\}$$

to record all queries K made by the adversary to random oracle R, along with the corresponding response $(r, \mathsf{K}') = R(\mathsf{K})$. The decryption oracle $\mathcal{O}_1$ uses this list as follows (boxed statements highlight changes to $\mathcal{O}_0$):

```
$\mathcal{O}_1(C)$

$\mathsf{K} \xleftarrow{\$} \mathsf{Dec}(\mathsf{sk}, C)$
$\boxed{\textbf{If } \nexists (r, \mathsf{K}') : (\mathsf{K}, (r, \mathsf{K}')) \in L_R \textbf{ then return } \perp}$
$(r, \mathsf{K}') = R(\mathsf{K})$
If $(C, \mathsf{K}) \neq \mathsf{Enc}(\mathsf{pk}; (r, \mathsf{K}))$ then return $\perp$
Return $\mathsf{K}'$
```

Note that Games 0 and 1 are perfectly indistinguishable, unless $\mathcal{A}$ ever outputs a ciphertext C with $\mathcal{O}_1(C) = \perp$, but $\mathcal{O}_0(C) \neq \perp$. Note that this happens if and only if $\mathcal{A}$ outputs C such that $C = \mathsf{Enc}(\mathsf{pk}; (r, \mathsf{K}))$, where r is the randomness defined by $(r, \mathsf{K}') = R(\mathsf{K})$, but without prior query of $R(\mathsf{K})$.

The random oracle R assigns a uniformly random value $r \in \{0,1\}^\rho$ to each query, so, by the γ-spreadness of PKEM, the probability that the ciphertext C output by the adversary "matches" the ciphertext produced by $\mathsf{Enc}(\mathsf{pk}; (r, \mathsf{K}))$ is $2^{-\gamma}$. Since $\mathcal{A}$ issues at most $q_\mathcal{O}$ queries to $\mathcal{O}_1$, this yields $\Pr[Q_1] \geq \Pr[Q_0] - q_\mathcal{O}/2^\gamma$.

Game 2. We make a minor conceptual modification. Instead of computing $(r, \mathsf{K}') = R(\mathsf{K})$ by evaluating R, $\mathcal{O}_2$ reads (r, K') from list L_R. More precisely:

```
$\mathcal{O}_2(C)$

$\mathsf{K} \xleftarrow{\$} \mathsf{Dec}(\mathsf{sk}, C)$
If $\nexists (r, \mathsf{K}') : (\mathsf{K}, (r, \mathsf{K}')) \in L_R$ then return $\bot$
Define $(r, \mathsf{K}')$ to be the unique tuple such that $(\mathsf{K}, (r, \mathsf{K}')) \in L_R$.
If $(C, \mathsf{K}) \neq \mathsf{Enc}(\mathsf{pk}; (r, \mathsf{K}))$ then return $\bot$
Return $\mathsf{K}'$
```

By definition of L_R it always holds that $(r, \mathsf{K}') = R(\mathsf{K})$ for all $(\mathsf{K}, (r, \mathsf{K}')) \in L_R$. Indeed (r, K'), is uniquely determined by K, because $(r, \mathsf{K}') = R(\mathsf{K})$ is a function. Since R is only evaluated by $\mathcal{O}_1$ if there exists a corresponding tuple $(\mathsf{K}, (r, \mathsf{K}')) \in L_R$ anyway, due to the changes introduced in Game 1, oracle $\mathcal{O}_2$ is equivalent to $\mathcal{O}_1$ and we have $\Pr[Q_2] = \Pr[Q_1]$.

Game 3. This game is identical to Game 2, except that whenever $\mathcal{A}$ queries a ciphertext C to oracle $\mathcal{O}_3$, then $\mathcal{O}_3$ first runs the $\mathsf{CheckPunct}$ algorithm associated to PKEM (cf. Definition 6). If $\mathsf{CheckPunct}(\mathsf{pk}, \mathcal{Q}, C) = \bot$, then it immediately returns $\bot$. Otherwise, it proceeds exactly like $\mathcal{O}_2$. More precisely:

```
$\mathcal{O}_3(C)$

If $\mathsf{CheckPunct}(\mathsf{pk}, \mathcal{Q}, C) = \bot$ then return $\bot$
$\mathsf{K} \xleftarrow{\$} \mathsf{Dec}(\mathsf{sk}, C)$
If $\nexists (r, \mathsf{K}') : (\mathsf{K}, (r, \mathsf{K}')) \in L_R$ then return $\bot$
Define $(r, \mathsf{K}')$ to be the unique tuple such that $(\mathsf{K}, (r, \mathsf{K}')) \in L_R$.
If $(C, \mathsf{K}) \neq \mathsf{Enc}(\mathsf{pk}; (r, \mathsf{K}))$ then return $\bot$
Return $\mathsf{K}'$
```

Recall that by public checkability (Definition 6) we have $\bot = \mathsf{Dec}(\mathsf{sk}, C) \iff \bot = \mathsf{CheckPunct}(\mathsf{pk}, \mathcal{Q}, C)$. Therefore the introduced changes are conceptual, and $\Pr[Q_3] = \Pr[Q_2]$.

Game 4. We modify the secret key used to decrypt the ciphertext. Let sk_0 denote the initial secret key generated by the experiment (that is, before any puncturing operation was performed). $\mathcal{O}_4$ uses sk_0 to compute $\mathsf{K} \xleftarrow{\$} \mathsf{Dec}(\mathsf{sk}_0, C)$ instead of $\mathsf{K} \xleftarrow{\$} \mathsf{Dec}(\mathsf{sk}, C)$, where sk is a possibly punctured secret key. More precisely:

```
$\mathcal{O}_4(C)$

If $\mathsf{CheckPunct}(\mathsf{pk}, \mathcal{Q}, C) = \perp$ then return $\perp$
$\mathsf{K} \xleftarrow{\$} \mathsf{Dec}(\mathsf{sk}_0, C)$
If $\nexists (r, \mathsf{K}') : (\mathsf{K}, (r, \mathsf{K}')) \in L_R$ then return $\perp$
Define $(r, \mathsf{K}')$ to be the unique tuple such that $(\mathsf{K}, (r, \mathsf{K}')) \in L_R$.
If $(C, \mathsf{K}) \neq \mathsf{Enc}(\mathsf{pk}; (r, \mathsf{K}))$ then return $\perp$
Return $\mathsf{K}'$
```

For indistinguishability from Game 3, we show that $\mathcal{O}_4(C) = \mathcal{O}_3(C)$ for all ciphertexts C. Let us first consider the case $\mathsf{Dec}(\mathsf{sk}, C) = \perp$. Then public checkability guarantees that $\mathcal{O}_4(C) = \mathcal{O}_3(C) = \perp$, due to the fact that $\mathsf{Dec}(\mathsf{sk}, C) = \perp \iff \mathsf{CheckPunct}(\mathsf{pk}, \mathcal{Q}, C) = \perp$.

Now let us consider the case $\mathsf{Dec}(\mathsf{sk}, C) \neq \perp$. In this case, the semi-correctness of punctured keys (3rd requirement of Definition 4) guarantees that $\mathsf{Dec}(\mathsf{sk}, C) = \mathsf{Dec}(\mathsf{sk}_0, C) = \mathsf{K} \neq \perp$.

After computing $\mathsf{Dec}(\mathsf{sk}_0, C)$, $\mathcal{O}_4$ performs exactly the same operations as $\mathcal{O}_3$ after computing $\mathsf{Dec}(\mathsf{sk}, C)$. Thus, in this case both oracles are perfectly indistinguishable, too. This yields that the changes introduced in Game 4 are purely conceptual, and we have $\Pr[Q_4] = \Pr[Q_3]$.

Remark. Due to the fact that we are now using the initial secret key to decrypt C, we have reached a setting where, due to the perfect correctness of the initial secret key sk_0, essentially a perfectly-correct encryption scheme is used – except that the decryption oracle implements a few additional abort conditions. Thus, we can now basically apply the standard Fujisaki-Okamoto transformation, but we must show that we are also able to simulate the additional abort imposed by the additional consistency checks properly. To this end, we first replace these checks with equivalent checks before applying the FO transformation.

Game 5. We replace the consistency checks performed by $\mathcal{O}_4$ with an equivalent check. More precisely, $\mathcal{O}_5$ works as follows:

```
$\mathcal{O}_5(C)$

If $\mathsf{CheckPunct}(\mathsf{pk}, \mathcal{Q}, C) = \perp$ then return $\perp$
$\mathsf{K} \xleftarrow{\$} \mathsf{Dec}(\mathsf{sk}_0, C)$
If $\nexists (r, \mathsf{K}') : ((\mathsf{K}, (r, \mathsf{K}')) \in L_R \wedge (C, \mathsf{K}) = \mathsf{Enc}(\mathsf{pk}; (r, \mathsf{K})))$ then return $\perp$
Return $\mathsf{K}'$ such that $(\mathsf{K}, (r, \mathsf{K}')) \in L_R \wedge (C, \mathsf{K}) = \mathsf{Enc}(\mathsf{pk}; (r, \mathsf{K}))$
```

This is equivalent, so that we have $\Pr[Q_5] = \Pr[Q_4]$.

Game 6. Observe that in Game 5 we check whether there exists a tuple (r, K') with $(\mathsf{K}, (r, \mathsf{K}')) \in L_R$ and $(C, \mathsf{K}) = \mathsf{Enc}(\mathsf{pk}; (r, \mathsf{K}))$, where K must match the secret key computed by $\mathsf{K} \xleftarrow{\$} \mathsf{Dec}(\mathsf{sk}_0, C)$.

In Game 6, we relax this check. We test only whether there exists any tuple $(\tilde{\mathsf{K}}, (\tilde{r}, \tilde{\mathsf{K}}')) \in L_R$ such that $(C, \tilde{\mathsf{K}}) = \mathsf{Enc}(\mathsf{pk}; (\tilde{r}, \tilde{\mathsf{K}})$ holds. Thus, it is not explicitly checked whether $\tilde{\mathsf{K}}$ matches the value $\mathsf{K} \xleftarrow{\$} \mathsf{Dec}(\mathsf{sk}_0, C)$. Furthermore, the corresponding value $\tilde{\mathsf{K}}'$ is returned. More precisely:

```
$\mathcal{O}_6(C)$

**If** $\mathsf{CheckPunct}(\mathsf{pk}, \mathcal{Q}, C) = \perp$ **then return** $\perp$
$\mathsf{K} \xleftarrow{\$} \mathsf{Dec}(\mathsf{sk}_0, C)$
**If** $\nexists(\tilde{r}, \tilde{\mathsf{K}}') : ((\tilde{\mathsf{K}}, (\tilde{r}, \tilde{\mathsf{K}}')) \in L_R \wedge (C, \tilde{\mathsf{K}}) = \mathsf{Enc}(\mathsf{pk}; (\tilde{r}, \tilde{\mathsf{K}})))$ **then return** $\perp$
**Return** $\tilde{\mathsf{K}}'$ such that $(\tilde{\mathsf{K}}, (\tilde{r}, \tilde{\mathsf{K}}')) \in L_R \wedge (C, \tilde{\mathsf{K}}) = \mathsf{Enc}(\mathsf{pk}; (\tilde{r}, \tilde{\mathsf{K}}))$
```

By the perfect correctness of the initial secret key sk_0, we have

$$(C, \tilde{\mathsf{K}}) = \mathsf{Enc}(\mathsf{pk}; (\tilde{r}, \tilde{\mathsf{K}})) \implies \mathsf{Dec}(\mathsf{sk}_0, C) = \tilde{\mathsf{K}},$$

so that we must have $\mathsf{K} = \tilde{\mathsf{K}}$. $\mathcal{O}_6$ is equivalent to $\mathcal{O}_5$, and $\Pr[Q_6] = \Pr[Q_5]$.

Game 7. This game is identical to Game 6, except that we change the decryption oracle again. Observe that the value K computed by $\mathsf{K} \xleftarrow{\$} \mathsf{Dec}(\mathsf{sk}_0, C)$ is never used by $\mathcal{O}_6$. Therefore the computation of $\mathsf{K} \xleftarrow{\$} \mathsf{Dec}(\mathsf{sk}_0, C)$ is obsolete, and we can remove it. More precisely, $\mathcal{O}_7$ works as follows.

```
$\mathcal{O}_7(C)$

**If** $\mathsf{CheckPunct}(\mathsf{pk}, \mathcal{Q}, C) = \perp$ **then return** $\perp$
**If** $\nexists(\tilde{r}, \tilde{\mathsf{K}}') : ((\tilde{\mathsf{K}}, (\tilde{r}, \tilde{\mathsf{K}}')) \in L_R \wedge (C, \tilde{\mathsf{K}}) = \mathsf{Enc}(\mathsf{pk}; (\tilde{r}, \tilde{\mathsf{K}})))$ **then return** $\perp$
**Return** $\tilde{\mathsf{K}}'$ such that $(\tilde{\mathsf{K}}, (\tilde{r}, \tilde{\mathsf{K}}')) \in L_R \wedge (C, \tilde{\mathsf{K}}) = \mathsf{Enc}(\mathsf{pk}; (\tilde{r}, \tilde{\mathsf{K}}))$
```

We have only removed an obsolete instruction, which does not change the output distribution of the decryption oracle. Therefore $\mathcal{O}_7$ simulates $\mathcal{O}_6$ perfectly, and we have $\Pr[Q_7] = \Pr[Q_6]$.

Reduction to OW-CPA-security. Now we are ready to describe the OW-CPA-adversary $\mathcal{B}$. $\mathcal{B}$ receives (pk, C^*). It samples a uniformly random key $\mathsf{K}' \xleftarrow{\$} \{0,1\}^\lambda$ and runs the IND-CCA-adversary $\mathcal{A}$ as a subroutine on input $(\mathsf{pk}, C^*, \mathsf{K}')$. Whenever $\mathcal{A}$ issues a Punc- or Corr-query, then $\mathcal{B}$ forwards this query to the OW-CPA-experiment and returns the response. In order to simulate the decryption oracle $\mathcal{O}$, adversary $\mathcal{B}$ implements the simulated oracle $\mathcal{O}_7$ from Game 7 described above. When $\mathcal{A}$ terminates, then $\mathcal{B}$ picks a uniformly random entry $(\hat{\mathsf{K}}, (\hat{r}, \hat{\mathsf{K}}')) \xleftarrow{\$} L_R$, and outputs $\hat{\mathsf{K}}$.

Analysis of the reduction. Let $\hat{Q}$ denote the event that $\mathcal{A}$ ever queries K_0 to random oracle R. Note that $\mathcal{B}$ siumulates Game 7 perfectly until Q_7 occurs, thus we have $\Pr[\hat{Q}] \geq \Pr[Q_7]$. Summing up, the probability that the value $\hat{\mathsf{K}}$ output by $\mathcal{B}$ matches the key encapsulated in C^* is therefore at least

$$\frac{\Pr[\hat{Q}]}{q_R} \geq \frac{\mathbf{Adv}^{\mathsf{IND\text{-}CCA}}_{\mathcal{A},\mathsf{PKEM}'}(\lambda, m, k) - q_{\mathcal{O}}/2^\gamma}{q_R}.$$

$\square$

Remark on the tightness. Alternatively, we could have based the security of our $\mathsf{IND\text{-}CCA}$-secure scheme on the $\mathsf{IND\text{-}CPA}$ (rather than $\mathsf{OW\text{-}CPA}$) security of PKEM'. In this case, we would have achieved a tighter reduction, as we would have been able to avoid guessing the index $(\hat{\mathsf{K}}, (\hat{r}, \hat{\mathsf{K}}')) \overset{\$}{\leftarrow} L_R$, at the cost of requiring stronger security of the underlying scheme.

From $\mathsf{IND\text{-}CCA}$-secure KEMs to $\mathsf{IND\text{-}CCA}$-secure encryption. It is well-known that one can generically transform an $\mathsf{IND\text{-}CCA}$-secure KEM into an $\mathsf{IND\text{-}CCA}$-secure encryption scheme, by combining it with a CCA-secure symmetric encryption scheme [16]. This construction applies to PKEMs as well.

2.7 Time-Based Bloom Filter Encryption

For a standard BFE scheme we have to update the public key after the secret key has been punctured n-times, because otherwise the false-positive probability would exceed an acceptable bound. In this section, we describe a construction of a scheme where the lifetime of the public key is split into *time slots*. Ciphertexts are associated with time slots, which assumes loosely synchronized clocks between sender and receiver of a ciphertext. The main advantage is that for a given bound on the correctness error, we are able to handle about the same number of puncturings *per time slot* as the basic scheme during the entire life time of the public key. We call this approach *time-based* Bloom filter encryption. It is inspired by the time-based approach used to construct puncturable encryption in [19,20], which in turn is inspired by the construction of forward-secret public-key encryption by Canetti, Halevi, and Katz [10].

Note that a time-based BFE scheme can trivially be obtained from any BFE scheme, by assigning an individual public/secret key pair for each time slot. However, if we want to split the life time of the public key into, say, 2^t time slots, then this would of course increase the size of keys by a factor 2^t. Since we want to enable a fine-grained use of time slots, to enable a very large number of puncturings over the entire lifetime of the public key without increasing the false positive probability beyond an unacceptable bound, we want to have 2^t as large as possible, but without increasing the size of the public key beyond an acceptable bound. To this end, we give a direct construction which increases the size of secret keys only by an *additive* amount of additional group elements, which is only *logarithmic* in the number of time slots. Thus, for 2^t time slots we have to add merely about t elements to the secret key, while the size of public keys remains even *constant*.

Formal definition. Likewise to considering our Bloom filter KEMs as an instantiation of a puncturable KEM with non-negligible correctness error, we can view the time-based approach analogously as an instantiation of a puncturable forward-secret KEM (PFSKEM) [20] with non-negligible correctness error. Consequently, we also chose to stick with the existing formal framework for PFSKEM, which we present subsequently. It is essentially our BFKEM Definition 2, augmented by time slots and an additional algorithm $\mathsf{PuncInt}$ that allows to puncture a secret key not with respect to a given ciphertext in a given time slot, but with respect to an entire time slot.

Definition 11 (PFSKEM [20]). *A puncturable forward-secret key encapsulation (PFSKEM) scheme is a tuple of the following* PPT *algorithms:*

$\mathsf{KGen}(1^\lambda, m, k, t)$: *Takes as input a security parameter λ, parameters m and k for the Bloom filter, and a parameter t specifying the number of time slots. It outputs a secret and public key $(\mathsf{sk}, \mathsf{pk})$, where we assume that the key-space $\mathcal{K}$ is implicit in* pk.

$\mathsf{Enc}(\mathsf{pk}, \tau)$: *Takes as input a public key* pk *and a time slot τ and outputs a ciphertext C and a symmetric key* K.

$\mathsf{PuncCtx}(\mathsf{sk}, \tau, C)$: *Takes as input a secret key* sk*, a time slot τ, a ciphertext C and outputs an updated secret key* sk'.

$\mathsf{Dec}(\mathsf{sk}, \tau, C)$: *Takes as input a secret key* sk*, a time slot τ, a ciphertext C and outputs a symmetric key* K *or $\perp$ if decapsulation fails.*

$\mathsf{PuncInt}(\mathsf{sk}, \tau)$: *Takes as input a secret key* sk*, a time slot τ and outputs an updated secret key* sk' *for the next slot $\tau + 1$.*

Due to the lack of space, we postpone the presentation of correctness, the additional properties (which are rather straightforward adaptions of the ones of a PKEM introduced in Section 2.3), as well as the IND-CPA/IND-CCA security notions to the full version.

Hierarchical IB-KEMs. We recall the basic definition of hierarchical identity-based key encapsulation schemes (HIB-KEMs) and their security.

Definition 12. *A $(t+1)$-level hierarchical identity-based key encapsulation scheme (*HIB-KEM*) with identity space $\mathcal{D} = \mathcal{D}_1 \times \cdots \times \mathcal{D}_{t+1}$, ciphertext space $\mathcal{C}$, and key space $\mathcal{K}$ consists of the following four algorithms:*

$\mathsf{HIBGen}(1^\lambda)$: *Takes as input a security parameter and outputs a key pair* $(\mathsf{mpk}, \mathsf{sk}_0)$. *We say that* mpk *is the master public key, and* sk_0 *is the* level-0 secret key.

$\mathsf{HIBDel}(\mathsf{sk}_{i-1}, d)$: *Takes as input a level-$i-1$ secret key* sk_{i-1} *with $i \in [t]$ and an element $d \in \mathcal{D}_i$ and outputs a level-i secret key* sk_i.

$\mathsf{HIBEnc}(\mathsf{mpk}, \boldsymbol{d})$: *Takes as input the master public key* mpk *and an identity $\boldsymbol{d} \in \mathcal{D}$ and outputs a ciphertext $C \in \mathcal{C}$ and a key* $\mathsf{K} \in \mathcal{K}$.

$\mathsf{HIBDec}(\mathsf{sk}_\ell, C)$: *Takes as input a level-t secret key* sk_t *and a ciphertext C, and outputs a value* $\mathsf{K} \in \mathcal{K} \cup \{\perp\}$*, where $\perp$ is a distinguished error symbol.*

Security definition. We will require only the very weak notion of one-wayness under selective-ID and chosen-plaintext attacks (OW-sID-CPA).

Definition 13 (OW-sID-CPA Security of HIB-KEM). *We define the advantage of an adversary $\mathcal{A}$ in the OW-sID-CPA experiment $\mathbf{Exp}^{\mathsf{OW\text{-}sID\text{-}CPA}}_{\mathcal{A},\mathsf{HIB\text{-}KEM}}(\lambda)$ as*

$$\mathbf{Adv}^{\mathsf{OW\text{-}sID\text{-}CPA}}_{\mathcal{A},\mathsf{HIB\text{-}KEM}}(\lambda) := \Pr\left[\mathbf{Exp}^{\mathsf{OW\text{-}sID\text{-}CPA}}_{\mathcal{A},\mathsf{HIB\text{-}KEM}}(\lambda) = 1\right].$$

We call a HIB-KEM *OW-sID-CPA secure, if $\mathbf{Adv}^{\mathsf{OW\text{-}sID\text{-}CPA}}_{\mathcal{A},\mathsf{HIB\text{-}KEM}}(\lambda)$ is a negligible function in λ for all PPT adversaries $\mathcal{A}$.*

```
$\mathbf{Exp}^{\text{OW-sID-CPA}}_{\mathcal{A},\text{HIB-KEM}}(\lambda)$
    $(\boldsymbol{d}^*, \mathtt{state}_{\mathcal{A}}) \overset{\$}{\leftarrow} \mathcal{A}(1^\lambda)$
    if $\boldsymbol{d}^* \notin \mathcal{D}$ return 0
    $(\mathsf{mpk}, \mathsf{sk}_0) \overset{\$}{\leftarrow} \mathsf{HIBGen}(1^\lambda)$, $(C, \mathsf{K}) \overset{\$}{\leftarrow} \mathsf{HIBEnc}(\mathsf{mpk}, \boldsymbol{d}^*)$
    $\mathsf{K}^* \overset{\$}{\leftarrow} \mathcal{A}(\mathsf{mpk}, C, \mathtt{state}_{\mathcal{A}})$
    return 1, if $\mathsf{K}^* = \mathsf{K}$
    return 0
```

Fig. 3. OW-sID-CPA security.

Time slots. We will construct a Bloom filter encryption scheme that allows to use 2^t time slots. We associate the i-th time slot with the string in $\{0,1\}^t$ that corresponds to the canonical t-bit binary representation of integer i.

Following [10,19,20], each time slot forms a leaf of an ordered binary tree of depth t. The root of the tree is associated with the empty string ϵ. We associate the left-hand descendants of the root with bit string 0, and the right-hand descendant with 1. Continuing this way, we associate the left descendant of node 0 with 00 and the right descendant with 01, and so on. We continue this procedure for all nodes, until we have constructed a complete binary tree of depth t. Note that two nodes at level t' of the tree are siblings if and only if their first $t'-1$ bits are equal, and that each bit string in $\{0,1\}^t$ is associated with a leaf of the tree. Note also that the tree is ordered, in the sense that the leftmost leaf is associated with 0^t, its right neighbour with $0^{t-1}1$, and so on.

Intuition of the construction. The basic idea behind the construction combines the binary tree approach of [10,19,20] with the Bloom filter encryption construction described in Section 2.5. We use a HIB-KEM with identity space

$$\mathcal{D} = \mathcal{D}_1 \times \cdots \times \mathcal{D}_{t+1} = \underbrace{\{0,1\} \times \cdots \times \{0,1\}}_{t \text{ times}} \times [m].$$

Each bit vector $\tau \in \mathcal{D}_1 \times \cdots \times \mathcal{D}_t = \{0,1\}^t$ corresponds to one time slot, and we set $\mathcal{D}_{t+1} = [m]$, where m is the size of the Bloom filter. The hierarchical key delegation property of the HIB-KEM enables the following features:

First, given a HIB-KEM key sk_τ for some "identity" (= time slot) $\tau \in \{0,1\}^t$, we can derive keys for all Bloom filter bits from sk_τ by computing

$$\mathsf{sk}_{\tau|d} \overset{\$}{\leftarrow} \mathsf{HIBDel}(\mathsf{sk}_\tau, d) \quad \text{for all} \quad d \in [m].$$

Second, in order to advance from time slot $\tau - 1$ to τ, we first compute

$$\mathsf{sk}_{\tau|d} \overset{\$}{\leftarrow} \mathsf{HIBDel}(\mathsf{sk}_\tau, d) \quad \text{for all} \quad d \in [m].$$

As soon as we have computed all Bloom filter keys for time slot τ, we "puncture" the tree "from left to right", such that we are able to compute all $\mathsf{sk}_{\tau'}$ with $\tau' > \tau$, but not any $sk_{\tau'}$ with $\tau' \leq \tau$. Here, we proceed exactly as in [10,19,20]. That is, in order to puncture at time slot τ, we first compute the HIB-KEM secret keys associated to all *right-hand siblings* of nodes that lie on the path from node τ to the root, and then we delete all secret keys associated to nodes that lie on the path from node τ to the root, including sk_τ itself. This yields a new secret key, which contains m level-$(t+1)$ HIB-KEM secret keys plus at most t HIB-KEM secret keys for levels $\leq t$, even though we allow for 2^t time slots.

Construction. Let $(\mathsf{HIBGen}, \mathsf{HIBDel}, \mathsf{HIBEnc}, \mathsf{HIBDec})$ be a HIB-KEM with key space $\mathcal{K}$ and identity space $\mathcal{D} = \mathcal{D}_1 \times \cdots \times \mathcal{D}_{t+1}$, where $\mathcal{D}_1 = \cdots = \mathcal{D}_t = \{0,1\}$, $\mathcal{D}_{t+1} = [m]$, and m is the size of the Bloom filter. Since we will only need selective security, one can instantiate such a HIB-KEM very efficiently, for example in bilinear groups based on the Boneh-Boyen-Goh [8] scheme, or based on lattices [1]. In the sequel, we will write $\{0,1\}^t$ shorthand for $\mathcal{D}_1 \times \cdots \times \mathcal{D}_t$, but keep in mind that the HIB-KEM supports more fine-grained key delegation. Let $\mathsf{B} = (\mathsf{BFGen}, \mathsf{BFUpdate}, \mathsf{BFCheck})$ be a Bloom filter for set $\{0,1\}^\lambda$. Furthermore, let $G' : \mathcal{K} \to \{0,1\}^\lambda$ be a hash function (which will be modeled as a random oracle [5] in the security proof).

We define $\mathsf{PKEM} = (\mathsf{KGen}, \mathsf{Enc}, \mathsf{PuncCtx}, \mathsf{Dec}, \mathsf{PuncInt})$ as follows.

$\underline{\mathsf{KGen}(1^\lambda, m, k, 2^t)}$: This algorithm first runs $((H_j)_{j\in[k]}, T) \xleftarrow{\$} \mathsf{BFGen}(m,k)$ to generate a Bloom filter, and $(\mathsf{mpk}, \mathsf{sk}_\epsilon) \xleftarrow{\$} \mathsf{HIBGen}(1^\lambda)$ to generate a key pair. Finally, the algorithm generates the keys for the first time slot. To this end, it first computes the HIB-KEM key for identity 0^t by recursively computing

$$\mathsf{sk}_{0^d} \xleftarrow{\$} \mathsf{HIBDel}(\mathsf{sk}_{0^{d-1}}, 0) \quad \text{for all} \quad d \in [t].$$

Then it computes the m Bloom filter keys for time slot 0^t by computing

$$\mathsf{sk}_{0^t|d} \xleftarrow{\$} \mathsf{HIBDel}(\mathsf{sk}_{0^t}, d) \quad \text{for all} \quad d \in [m],$$

and setting $\mathsf{sk}_{\mathsf{Bloom}} := (\mathsf{sk}_{0^t|d})_{d\in[m]}$. Finally, it punctures the secret key sk_ϵ at position 0^t, by computing

$$\mathsf{sk}_{0^{d-1}1} \xleftarrow{\$} \mathsf{HIBDel}(\mathsf{sk}_{0^{d-1}}, 1) \quad \text{for all} \quad d \in [t],$$

and setting $\mathsf{sk}_{\mathsf{time}} := (\mathsf{sk}_{0^{d-1}|1})_{d\in[t]}$. The algorithm outputs

$$\mathsf{sk} := (T, \mathsf{sk}_{\mathsf{Bloom}}, \mathsf{sk}_{\mathsf{time}}) \text{ and } \mathsf{pk} := (\mathsf{mpk}, (H_j)_{j\in[k]}).$$

$\underline{\mathsf{Enc}(\mathsf{mpk}, \tau)}$: On input mpk and time slot identifier $\tau \in \{0,1\}^t$, this algorithm first samples a random string $c \xleftarrow{\$} \{0,1\}^\lambda$ and a random key $\mathsf{K} \xleftarrow{\$} \{0,1\}^\lambda$. Then it defines k HIB-KEM identities as $\boldsymbol{d}_j := (\tau, H_j(c)) \in \mathcal{D}$ for $j \in [k]$, and generates k HIB-KEM key encapsulations as

$$(C_j, \mathsf{K}_j) \xleftarrow{\$} \mathsf{HIBEnc}(\mathsf{mpk}, \boldsymbol{d}_j) \quad \text{for} \quad j \in [k].$$

Finally, it outputs the ciphertext $C := (c, (C_j, G'(\mathsf{K}_j) \oplus \mathsf{K})_{j\in[k]})$.

Note that the ciphertexts essentially consists of $k+1$ elements of $\{0,1\}^\lambda$, plus k elements of $\mathcal{C}$, where k is the Bloom filter parameter.

$\underline{\mathsf{PuncCtx}(\mathsf{sk}, C)}$: Given a ciphertext $C := (c, (C_j, G'(\mathsf{K}_j) \oplus \mathsf{K})_{j\in[k]})$, and secret key $\mathsf{sk} = (T, \mathsf{sk}_{\mathsf{Bloom}}, \mathsf{sk}_{\mathsf{time}})$ where $\mathsf{sk}_{\mathsf{Bloom}} = (\mathsf{sk}_{\tau|d})_{d\in[m]}$, the puncturing algorithm first computes $T' = \mathsf{BFUpdate}((H_j)_{j\in[k]}, T, c)$. Then, for each $i \in [m]$, it defines

$$\mathsf{sk}'_{\tau|i} := \begin{cases} \mathsf{sk}_{\tau|i} & \text{if } T'[i] = 0, \text{ and} \\ \perp & \text{if } T'[i] = 1, \end{cases}$$

where $T'[i]$ denotes the i-th bit of T'. Finally, this algorithm sets $\mathsf{sk}'_{\mathsf{Bloom}} = (\mathsf{sk}'_{\tau|d})_{d\in[m]}$ and returns $\mathsf{sk}' = (T', \mathsf{sk}'_{\mathsf{Bloom}}, \mathsf{sk}_{\mathsf{time}})$.

Remark. We note again that the above procedure is correct even if the procedure is applied repeatedly, with the same arguments as for the construction from Section 2.5. Also, the puncturing algorithm essentially only evaluates k universal hash functions and then deletes a few secret keys, which makes this procedure extremely efficient.

$\underline{\mathsf{Dec}(\mathsf{sk}, C)}$: Given $\mathsf{sk} = (T, \mathsf{sk}_{\mathsf{Bloom}}, \mathsf{sk}_{\mathsf{time}})$ where $\mathsf{sk}_{\mathsf{Bloom}} = (\mathsf{sk}_{\tau|d})_{d\in[m]}$ and ciphertext $C := (c, (C_j, G_j)_{j\in[k]})$. If $\mathsf{sk}_{\tau|H_j(c)} = \perp$ for all $j \in [k]$, then it outputs $\perp$. Otherwise, it picks the smallest index j such that $\mathsf{sk}_{\tau|H_j(c)} \neq \perp$, computes

$$\mathsf{K}_j = \mathsf{HIBDec}(\mathsf{sk}_{\tau|H_j(c)}, C_j),$$

and returns $\mathsf{K} = G_j \oplus G'(\mathsf{K}_j)$.

Remark. Again we have $\mathsf{Dec}(\mathsf{sk}, C) \neq \perp \iff \mathsf{BFCheck}(H, T, c) = 0$, which guarantees extended correctness in the sense of Definition 4.

$\underline{\mathsf{PuncInt}(\mathsf{sk}, \tau)}$: Given a secret key $\mathsf{sk} = (T, \mathsf{sk}_{\mathsf{Bloom}}, \mathsf{sk}_{\mathsf{time}})$ for time interval $\tau' < \tau$, the time puncturing algorithm proceeds as follows. First, it resets the Bloom filter by setting $T := 0^m$. Then it uses the key delegation algorithm to first compute sk_τ. This key can be computed from the keys contained in $\mathsf{sk}_{\mathsf{time}}$, because sk is a key for time interval $\tau' < \tau$. Then it computes

$$\mathsf{sk}_{\tau|d} \xleftarrow{\$} \mathsf{HIBDel}(\mathsf{sk}_\tau, d) \quad \text{for all} \quad d \in [m],$$

and redefines $\mathsf{sk}_{\mathsf{Bloom}} := (\mathsf{sk}_{\tau|d})_{d\in[m]}$. Finally, it updates $\mathsf{sk}_{\mathsf{time}}$ by computing the HIB-KEM secret keys associated to all *right-hand siblings* of nodes that lie on the path from node τ to the root and adds the corresponding keys to $\mathsf{sk}_{\mathsf{time}}$. Then it deletes all keys from $\mathsf{sk}_{\mathsf{time}}$ that lie on the path from τ to the root.

Remark. Note that puncturing between time intervals may become relatively expensive. Depending on the choice of Bloom filter parameters, in particular on m, this may range between 2^{15} and 2^{25} HIBE key delegations. However, the main advantage of Bloom filter encryption over previous constructions of puncturable encryption is that these computations must not be performed "online", during puncturing, but can actually be computed separately (for instance, parallel on a different computer, or when a server has low workload, etc.).

Correctness error of this scheme. With exactly the same arguments as for the scheme from Section 2.5, one can verify that the correctness error of this scheme is essentially identical to the false positive probability of the Bloom filter, unless a given ciphertext $C = (c, (C_j, G_j)_{j \in [k]})$ has a value of c which is identical to the value of c of any previous ciphertext. Since c is uniformly random in $\{0,1\}^\lambda$, this probability is approximately $2^{-k} + n \cdot 2^{-\lambda}$.

Extended correctness. It is straightforward to verify that the scheme satisfies extended correctness in the sense of Definition 4.

CPA Security. Below we state theorem for CPA security of our scheme.

Theorem 4. *From each efficient adversary $\mathcal{B}$ that issues q queries to random oracle G' we can construct an efficient adversary $\mathcal{A}$ with*

$$\mathbf{Adv}^{\mathsf{OW\text{-}sID\text{-}CPA}}_{\mathcal{A},\mathsf{HIB\text{-}KEM}}(\lambda) \geq \frac{\mathbf{Adv}^{\mathsf{s\text{-}CPA}}_{\mathcal{B},\mathsf{PFSKEM}}(\lambda, m, k)}{qk}.$$

The proof is almost identical to the proof of Theorem 1 and a straightforward reduction to the security of the underlying HIB-KEM. We sketch it in the full version.

CCA Security. In order to apply the Fujisaki-Okamoto [16] transform in the same way as done in Section 2.6 to achieve CCA security, we need to show that the time based variants of the properties presented in Section 2.3 are satisfied (for the formal definitions of those properties we refer the reader to the full version). First, using a full-blown HIBE as a starting point yields a separable HIB-KEM as discussed in Section 2.3. Hence, the separable randomness is satisfied. Moreover, the publicly-checkable puncturing is given by construction (as in Section 2.5). Regarding extended correctness, the impossibility of false-negatives is given by construction, the perfect correctness of the non-punctured secret key is given by the perfect correctness of the HIBE and the semi-correctness of punctured secret keys is given by construction. Finally, γ-spreadness is also given by construction: the ciphertext component c is chosen uniformly at random from $\{0,1\}^\lambda$. Consequently, all properties are satisfied. We note that one could omit c in the ciphertext if the concretely used HIBE ciphertexts are already sufficiently random. Considering the HIBE of Boneh-Boyen-Goh [8], HIBE ciphertexts are of the form $(g^r, (h_1^{I_1} \cdots h_t^{I_t} \cdot h_0)^r, H(e(g_1, g_2)^r) \oplus \mathsf{K})$, for honestly generated fixed group elements $g, g_1, g_2, h_0, \ldots, h_t$, universal hash function H, fixed K and fixed integers $I_1, \ldots, I_t$. Consequently, we have that the ciphertext has at least min-entropy $\log_2 p$ with p being the order of the groups. We want to mention that also many other HIBE construction satisfy the required properties, including, for example [17,30,12].

Remark on CCA Security. Alternatively to applying the FO transform to a PFSKEM satsifying the additional properties of extended correctness, seperable randomness, publicly checkable puncturing and γ-spreadness to obtain CCA security, we can add another HIBE level to obtain IND-CCA security via the CHK transform [10] in the standard model, and thus to avoid random oracles if required.

3 Forward-Secret 0-RTT Key Exchange

In [20], GHJL provide a formal model for forward-secret one-pass key exchange (FSOPKE) by extending the one-pass key exchange [22] by Halevi and Krawczyk. They provide a security model for FSOPKE which requires both forward secrecy and replay protection from the FSOPKE protocol and captures unilateral authentication of the server and mutual authentication simultaneously. We recap the definition of FSOPKE with a slightly adapted correctness notion in the full version.

Construction. The construction in [20] builds on puncturable forward-secret key encapsulation (PFSKEM), and we can now directly plug our construction of time-based BFE (PFSKEM) as defined in Def. 11 into the construction of [20, Def. 12], yielding a forward-secret 0-RTT key exchange protocols with non-negligible correctness error:

$\underline{\mathsf{FSOPKE.KGen}(1^\lambda, r, \tau_{max})}$: Outputs $(\mathsf{pk}, \mathsf{sk})$ as follows: if $r = \mathsf{server}$, then obtain $(PK, SK) \leftarrow \mathsf{KGen}(1^\lambda, m, k, t)$ (for suitable choices of m, k and t) and set $\mathsf{pk} := (PK, \tau_{max})$ and $\mathsf{sk} := (SK, \tau, \tau_{max})$, for $\tau := 1$. If $r = \mathsf{client}$, then set $(\mathsf{pk}, \mathsf{sk}) := (\perp, \tau)$, for $\tau := 1$.

$\underline{\mathsf{FSOPKE.RunC}(\mathsf{sk}, \mathsf{pk})}$: Outputs $(\mathsf{sk}', \mathsf{K}, M)$ as follows: for $\mathsf{sk} = \tau$ and $\mathsf{pk} = (PK, \tau_{max})$, if $\tau > \tau_{max}$, then set $(\mathsf{sk}', \mathsf{K}, M) := (\mathsf{sk}, \perp, \perp)$, otherwise obtain $(C, \mathsf{K}) \leftarrow \mathsf{Enc}(\mathsf{pk}, \tau)$ and set $(\mathsf{sk}', \mathsf{K}, M) := (\tau + 1, \mathsf{K}, C)$.

$\underline{\mathsf{FSOPKE.RunS}(\mathsf{sk}, \mathsf{pk}, M)}$: Outputs $(\mathsf{sk}', \mathsf{K})$ as follows: for $\mathsf{sk} = (SK, \tau, \tau_{max})$ and $\mathsf{pk} = \perp$, if $SK = \perp$ or $\tau > \tau_{max}$, then set $(\mathsf{sk}', \mathsf{K}) := (\mathsf{sk}, \perp)$ and abort. Obtain $\mathsf{K} \leftarrow \mathsf{Dec}(SK, \tau, M)$. If $\mathsf{K} = \perp$, then set $(\mathsf{sk}', \mathsf{K}) = (\mathsf{sk}, \perp)$, otherwise obtain $SK' \leftarrow \mathsf{PuncCtx}(SK, \tau, M)$ and set $(\mathsf{sk}', \mathsf{K}) = ((SK', \tau, \tau_{max}), \mathsf{K})$.

$\underline{\mathsf{FSOPKE.TimeStep}(\mathsf{sk}, r)}$: Outputs sk' as follows: if $r = \mathsf{server}$, then for $\mathsf{sk} = (SK, \tau, \tau_{max})$: if $\tau \geq \tau_{max}$, then set $\mathsf{sk}' := (\perp, \tau + 1, \tau_{max})$ and abort, otherwise obtain $SK' \leftarrow \mathsf{PuncInt}(SK, \tau)$ and set $\mathsf{sk}' := (SK', \tau + 1, \tau_{max})$ and abort. If $r = \mathsf{client}$, then for $sk = \tau$, set $\mathsf{sk}' := \tau + 1$.

Correctness of the FSOPKE follows from the (extended) correctness property of the underlying PFSKEM and security guarantees hold due to [20, Theorem 2]. We state the following corollary:

Corollary 1. *When instantiated with the PFSKEM from Section 2.7, the above FSOPKE construction is a correct and secure FSOPKE protocol (with unilateral authentication).*

3.1 Analysis

In Table 1, we provide an overview of all existing practically instantiable approaches to construct forward-secret (time-based) PKEM with the one proposed in this paper.[6] We compare all schemes for an arbitrary number ℓ of time slots,

[6] We consider all but the PE schemes from indistinguishability obfuscation [13,11].

where for sake of simplicity we assume $\ell = 2^w$ for some integer w, (corresponding to our time-based BFE/BFKEM) and only count the expensive cryptographic operations, i.e., such as group exponentiations and pairings.

Scheme	$|\mathsf{pk}|$	$|\mathsf{sk}|$	$|C|$	Dec	PuncCtx	PuncInt
		$\ell = 2^w$ time slots (PFSKEM)				
GM	$(w+5)|\mathbb{G}_1|$	$(2w+3p+5)|\mathbb{G}_2|$	$3|\mathbb{G}_1| + |\mathbb{G}_T|$	$O(p)$	$O(1)$	$O(w^2)$
GHJL	$(w+35)|\mathbb{G}_2|$	$\leq 3(p \cdot 2\lambda + w)|\mathbb{G}_2|$	$6|\mathbb{G}_1| + 2|\mathbb{Z}_p|$	$O(\lambda^2)$	$O(\lambda^2)$	$O(w^2)$
Ours	$(w+4)|\mathbb{G}_2|$	$(2me^{-kp/m} + w(2+w))|\mathbb{G}_2|$	$2|\mathbb{G}_1| + (4k+2)\lambda$	$O(k)$	$O(k)$	$O(w^2+m)$

Table 1. Overview of the existing approaches to PFSKEM. We denote by p the number a secret key is already punctured, and ℓ the maximum number of time slots. We consider the GHJL [20] instantiation with the BKP-HIBE of [6], the GM [19] and our instantiations with the BBG-HIBE [8], though other HIBE schemes may lead to different parameters. Finally, note that $p \leq 2^{20}$, k and m refer to the parameters in the Bloom filter, where k is some orders of magnitude smaller than λ, i.e., $k = 10$ vs. $\lambda = 128$, and $|\mathbb{G}_i|$ denotes the bitlength of an element from $\mathbb{G}_i$.

To quickly summarize the schemes: The most interesting characteristic of our approach compared to previous approaches is that our scheme allows to offload all expensive operation to an offline phase, i.e., to the puncturing of time intervals. Here, in addtion to the $O(w^2)$ operations which are common to all existing approaches, we have to generate a number of keys, linear in the size m of the Bloom filter. We believe that accepting this additional overhead in favor of blazing fast online puncturing and decryption operations is a viable tradeoff. For the online phase, our approach has a ciphertext size depending on k (where $k = 10$ is a reasonable choice), decryption depends on k, the secret key shrinks with increasing amount of puncturings and one does only require to securely delete secret keys during puncturing (note that all constructions have to implement a secure-delete functionality for secret keys within puncturing anyways). In contrast, decryption and puncturing in GHJL is highly inefficient and takes several seconds to minutes on decent hardware for reasonable deployment parameters as it involves a large amount of $O(\lambda^2)$ HIBE delegations and consequently expensive group operations. In the GM scheme[7], puncturing is efficient, but the size of the secret key and thus cost of decryption grows in the number of puncturings p. Hence, it gets impractical very soon. More precisely, cost of decryption requires a number of pairing evaluations that depends on the number of puncturings, and can be in the order of 2^{20} for realistic deployment parameters.

[7] Although GM supports an arbitrary number d of tags in a ciphertext, we consider the scheme with only using a single tag (which is actually favourable for the scheme) to be comparable to GHJL as well as our approach.

4 Conclusion

In this paper we introduced the new notion of Bloom filter encryption (BFE) as a variant of puncturable encryption which tolerates a non-negligible correctness error. We presented various BFKEM constructions. The first one is a simple and very efficient construction which builds upon ideas known from the Boneh-Franklin IBE. The second one, which is presented in the full version, is a generic construction from CP-ABEs which achieves constant size ciphertexts. Furthermore, we extended the notion of BFE to the forward-secrecy setting and also presented a construction of what we call a time-based BFE (TB-BFE). This construction is based on HIBEs and in particular can be instantiated very efficiently using the Boneh-Boyen-Goh Tiny HIBE [8]. Our time-based BFKEM can directly be used to instantiate forward-secret 0-RTT key exchange (fs 0-RTT KE) as in [20].

From a practical viewpoint, our motivation stems from the observation that forward-secret 0-RTT KE requires very efficient decryption and puncturing. Our framework—for the first time—allows to realize practical forward-secret 0-RTT KE, even for larger server loads: while we only require to delete secret keys upon puncturing, puncturing in [20] requires, besides deleting secret-key components, additional computations in the order of seconds to minutes on decent hardware. Likewise, when using [19] in the forward-secret 0-RTT KE protocol given in [20], one requires computations in the order of the current number of puncturings upon decryption, while we achieve decryption to be independent of this number. Finally, we believe that BFE will find applications beyond forward-secret 0-RTT KE protocols.

Acknowledgments. This research was supported by H2020 project Prismacloud, grant agreement n°644962, H2020 project Credential, grant agreement n°653454, and the German Research Foundation (DFG), project JA 2445/2-1. We thank Kai Gellert and all anonymous reviewers for their valuable comments.

References

1. 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, Germany, French Riviera (May 30 – Jun 3, 2010)
2. Barbulescu, R., Duquesne, S.: Updating key size estimations for pairings. Cryptology ePrint Archive, Report 2017/334 (2017), http://eprint.iacr.org/2017/334
3. Barreto, P.S.L.M., Lynn, B., Scott, M.: Constructing elliptic curves with prescribed embedding degrees. In: Cimato, S., Galdi, C., Persiano, G. (eds.) SCN 02. LNCS, vol. 2576, pp. 257–267. Springer, Heidelberg, Germany, Amalfi, Italy (Sep 12–13, 2003)
4. Barreto, P.S.L.M., Naehrig, M.: Pairing-friendly elliptic curves of prime order. In: Preneel, B., Tavares, S. (eds.) SAC 2005. LNCS, vol. 3897, pp. 319–331. Springer, Heidelberg, Germany, Kingston, Ontario, Canada (Aug 11–12, 2006)

5. Bellare, M., Rogaway, P.: Random oracles are practical: A paradigm for designing efficient protocols. In: Ashby, V. (ed.) ACM CCS 93. pp. 62–73. ACM Press, Fairfax, Virginia, USA (Nov 3–5, 1993)
6. Blazy, O., Kiltz, E., Pan, J.: (Hierarchical) identity-based encryption from affine message authentication. In: Garay, J.A., Gennaro, R. (eds.) CRYPTO 2014, Part I. LNCS, vol. 8616, pp. 408–425. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 17–21, 2014)
7. Bloom, B.H.: Space/time trade-offs in hash coding with allowable errors. Commun. ACM 13(7), 422–426 (1970)
8. 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, Germany, Aarhus, Denmark (May 22–26, 2005)
9. Boneh, D., Franklin, M.K.: Identity-based encryption from the Weil pairing. In: Kilian, J. (ed.) CRYPTO 2001. LNCS, vol. 2139, pp. 213–229. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 19–23, 2001)
10. 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, Germany, Warsaw, Poland (May 4–8, 2003)
11. Canetti, R., Raghuraman, S., Richelson, S., Vaikuntanathan, V.: Chosen-ciphertext secure fully homomorphic encryption. In: Public-Key Cryptography - PKC 2017. pp. 213–240 (2017)
12. Chen, J., Wee, H.: Fully, (almost) tightly secure IBE and dual system groups. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013, Part II. LNCS, vol. 8043, pp. 435–460. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 18–22, 2013)
13. Cohen, A., Holmgren, J., Nishimaki, R., Vaikuntanathan, V., Wichs, D.: Watermarking cryptographic capabilities. In: Wichs, D., Mansour, Y. (eds.) 48th ACM STOC. pp. 1115–1127. ACM Press, Cambridge, MA, USA (Jun 18–21, 2016)
14. Derler, D., Krenn, S., Lorünser, T., Ramacher, S., Slamanig, D., Striecks, C.: Revisiting proxy re-encryption: Forward secrecy, improved security, and applications. In: Abdalla, M. (ed.) PKC 2018. LNCS, Springer (2018)
15. Dierks, T.: The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246 (Aug 2008), https://rfc-editor.org/rfc/rfc5246.txt
16. Fujisaki, E., Okamoto, T.: Secure integration of asymmetric and symmetric encryption schemes. In: Wiener, M.J. (ed.) CRYPTO'99. LNCS, vol. 1666, pp. 537–554. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 15–19, 1999)
17. Gentry, C., Silverberg, A.: Hierarchical ID-based cryptography. In: Zheng, Y. (ed.) ASIACRYPT 2002. LNCS, vol. 2501, pp. 548–566. Springer, Heidelberg, Germany, Queenstown, New Zealand (Dec 1–5, 2002)
18. Goyal, V., Pandey, O., Sahai, A., Waters, B.: Attribute-based encryption for fine-grained access control of encrypted data. In: Juels, A., Wright, R.N., Vimercati, S. (eds.) ACM CCS 06. pp. 89–98. ACM Press, Alexandria, Virginia, USA (Oct 30 – Nov 3, 2006), available as Cryptology ePrint Archive Report 2006/309
19. 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, San Jose, CA, USA (May 17–21, 2015)
20. Günther, F., Hale, B., Jager, T., Lauer, S.: 0-RTT key exchange with full forward secrecy. In: Coron, J., Nielsen, J.B. (eds.) EUROCRYPT 2017, Part II. LNCS, vol. 10211, pp. 519–548. Springer, Heidelberg, Germany, Paris, France (May 8–12, 2017)

21. Hale, B., Jager, T., Lauer, S., Schwenk, J.: Simple security definitions for and constructions of 0-RTT key exchange. In: Gollmann, D., Miyaji, A., Kikuchi, H. (eds.) ACNS 17. LNCS, vol. 10355, pp. 20–38. Springer, Heidelberg, Germany, Kanazawa, Japan (Jul 10–12, 2017)
22. Halevi, S., Krawczyk, H.: One-pass HMQV and asymmetric key-wrapping. In: Catalano, D., Fazio, N., Gennaro, R., Nicolosi, A. (eds.) PKC 2011. LNCS, vol. 6571, pp. 317–334. Springer, Heidelberg, Germany, Taormina, Italy (Mar 6–9, 2011)
23. Hofheinz, D., Hövelmanns, K., Kiltz, E.: A modular analysis of the fujisaki-okamoto transformation. Cryptology ePrint Archive, Report 2017/604 (2017), http://eprint.iacr.org/2017/604
24. Kim, T., Barbulescu, R.: Extended tower number field sieve: A new complexity for the medium prime case. In: Robshaw, M., Katz, J. (eds.) CRYPTO 2016, Part I. LNCS, vol. 9814, pp. 543–571. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 14–18, 2016)
25. Menezes, A., Sarkar, P., Singh, S.: Challenges with assessing the impact of NFS advances on the security of pairing-based cryptography. IACR Cryptology ePrint Archive 2016, 1102 (2016), http://eprint.iacr.org/2016/1102
26. Naor, M., Yogev, E.: Bloom filters in adversarial environments. In: Gennaro, R., Robshaw, M.J.B. (eds.) CRYPTO 2015, Part II. LNCS, vol. 9216, pp. 565–584. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 16–20, 2015)
27. Ostrovsky, R., Sahai, A., Waters, B.: Attribute-based encryption with non-monotonic access structures. In: Ning, P., di Vimercati, S.D.C., Syverson, P.F. (eds.) ACM CCS 07. pp. 195–203. ACM Press, Alexandria, Virginia, USA (Oct 28–31, 2007)
28. Rescorla, E.: The Transport Layer Security (TLS) Protocol Version 1.3. Internet-Draft draft-ietf-tls-tls13-20, Internet Engineering Task Force (Apr 2017), https://datatracker.ietf.org/doc/html/draft-ietf-tls-tls13-20, work in Progress
29. Thomson, M., Iyengar, J.: QUIC: A UDP-Based Multiplexed and Secure Transport. Internet-Draft draft-ietf-quic-transport-02, Internet Engineering Task Force (Mar 2017), https://datatracker.ietf.org/doc/html/draft-ietf-quic-transport-02, work in Progress
30. Waters, B.: Dual system encryption: Realizing fully secure IBE and HIBE under simple assumptions. In: Halevi, S. (ed.) CRYPTO 2009. LNCS, vol. 5677, pp. 619–636. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 16–20, 2009)
31. Wu, D.J., Taly, A., Shankar, A., Boneh, D.: Privacy, discovery, and authentication for the internet of things. In: Askoxylakis, I.G., Ioannidis, S., Katsikas, S.K., Meadows, C.A. (eds.) ESORICS 2016, Part II. LNCS, vol. 9879, pp. 301–319. Springer, Heidelberg, Germany, Heraklion, Greece (Sep 26–30, 2016)