Puncturable Key Wrapping and Its Applications

Matilda Backendal, Felix Günther, and Kenneth G. Paterson

Department of Computer Science, ETH Zurich, Zurich, Switzerland
{mbackendal,kenny.paterson}@inf.ethz.ch, mail@felixguenther.info

December 4, 2022

Abstract

We introduce *puncturable key wrapping* (PKW), a new cryptographic primitive that supports fine-grained forward security properties in symmetric key hierarchies. We develop syntax and security definitions, along with provably secure constructions for PKW from simpler components (AEAD schemes and puncturable PRFs). We show how PKW can be applied in two distinct scenarios. First, we show how to use PKW to achieve forward security for TLS 1.3 0-RTT session resumption, even when the server's long-term key for generating session tickets gets compromised. This extends and corrects a recent work of Aviram, Gellert, and Jager (Journal of Cryptology, 2021). Second, we show how to use PKW to build a protected file storage system with file shredding, wherein a client can outsource encrypted files to a potentially malicious or corrupted cloud server whilst achieving strong forward-security guarantees, relying only on local key updates.

A preliminary version of this paper appears in the proceedings of the *28th International Conference on the Theory and Application of Cryptology and Information Security (ASIACRYPT 2022)*. This is the full version.

Contents

1 Introduction

Key wrapping. Key encryption, or *key wrapping*, is a mechanism often deployed to build symmetric key hierarchies: systems in which the confidentiality and integrity of multiple cryptographic keys are protected by a single (master wrapping) key. The wrapped keys may in turn be used to secure data at a more fine-grained level, e.g., at the level of individual files, messages, or financial transactions. This hierarchical approach eases key management: it allows strong but more expensive protection to be applied to a small number of wrapping keys while limiting the security impact if individual wrapped keys are exposed. Key wrapping is widely used in practice; specific schemes have been standardized by NIST in [27]. Formal foundations for key wrapping were established in [51].

As a pertinent example, when using the pre-shared key (PSK) mode of TLS 1.3 [49] for session resumption, new sessions between client and server are protected by independent, symmetric keys (denoted PSK) established in an earlier session. To reduce storage overhead, servers often use a long-term symmetric encryption key to *wrap* PSKs into so-called *tickets*. These tickets are sent to the client, thereby outsourcing the PSK storage from the server to the client.

Another example of key hierarchies is found in cloud storage systems, where service providers encrypt data before storing it on their servers—so called *encryption at rest*. The encryption is done to meet customer demand and regulatory requirements. To ensure good key-hygiene, best practices stipulate that separate encryption keys be used for separate files (or even parts of large files). To this end, cloud storage providers use a new *data encryption key* (DEK) to encrypt each (part of a) file. The DEK is then wrapped using a *key encryption key* (KEK) and stored together with the encrypted file. Here, using a key hierarchy also allows for a form of *key rotation*, a process in which a key is replaced by a fresh one, and the encrypted data is updated to be secured under the new key. The technique used by all four of Amazon Web Services [4], Google Cloud [34], IBM Cloud [38] and Microsoft Azure [46] is to rotate only the KEK rather than all of the DEKs. This limits the amount of data that needs to be re-encrypted under the new KEK to just the DEKs that were wrapped under the original KEK, rather than the actual files themselves. This approach provides an efficient but security-limited form of key rotation [28].

Forward-secure session resumption and puncturable encryption. Aviram, Gellert, and Jager (AGJ) [1, 2] observed that the key hierarchy induced by the ticketing mechanism in TLS 1.3 PSK mode can be used to achieve *forward security* for resumed sessions. By updating the Session Ticket Encryption Key (STEK) after accepting the ticket of a resumed session, and deleting the corresponding PSK, the confidentiality of the session is guaranteed even against an attacker who later compromises the STEK. AGJ formalized this idea with their notion of a forward-secure *session resumption protocol*. The per-session forward security enjoyed by such a resumption protocol is reminiscent of the fine-grained forward security achieved by *puncturable encryption* [35], and indeed, AGJ make use of *puncturable* pseudo-random functions (PPRFs) [13, 17, 40] for their construction. Their innovation naturally begs the question: Can puncturing be combined with key hierarchies to bring fine-grained forward security also to other applications? This work provides the affirmative response.

1.1 Our Contributions

We investigate how puncturing can be combined with key wrapping to provide fine-grained forward security in applications using a symmetric key hierarchy. To this end, we introduce a new cryptographic primitive that we call *puncturable key wrapping* (PKW). We provide formal definitions, relations between security notions, and an efficient, generic construction for PKW. We also show how to use PKW in two sample applications: TLS ticketing (inspired by [2], but addressing several shortcomings of that work) and protected file storage. We argue that, while PKW is closely related to existing primitives like PPRFs, it provides a useful abstraction that more intuitively captures what is needed for achieving fine-grained forward security in symmetric key hierarchies. This makes building applications conceptually simpler and less error-prone.[1]

Puncturable key wrapping. A puncturable key-wrapping scheme provides the basic functionality needed for a symmetric key hierarchy: algorithms to wrap and unwrap data encryption keys under a master secret key. Additionally, a puncturing algorithm allows the master secret key to be updated such that specific wrapped data encryption keys are rendered irrecoverable. Our PKW syntax merges classical key wrapping/deterministic authenticated encryption [51] with tag-based puncturable encryption [35].

[1] A broad analogy that readers may find useful: PKW is to PPRFs as AEAD is to block ciphers.

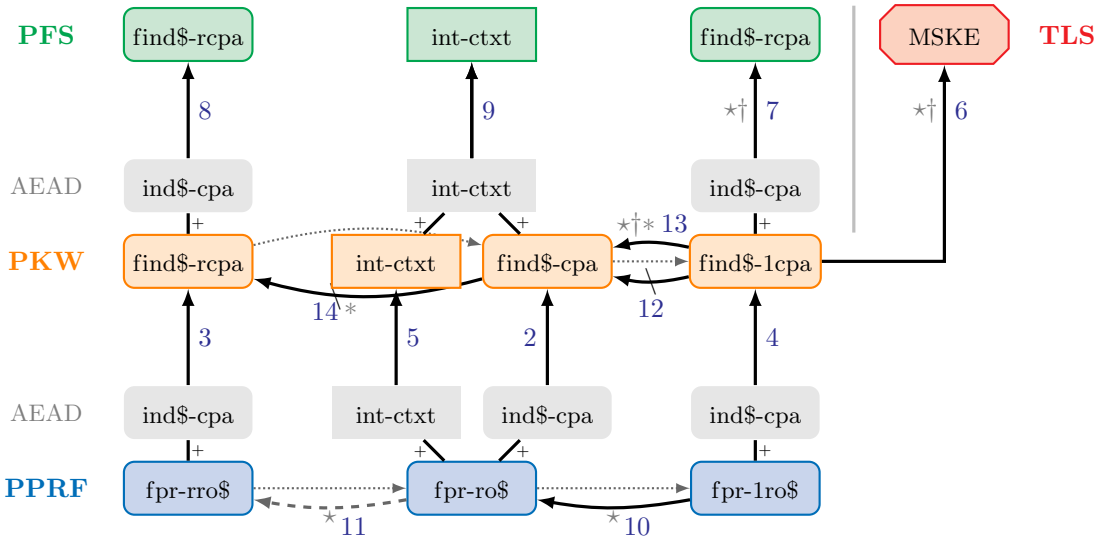

Figure 1: Security notions and relations for PPRFs, puncturable key-wrapping (PKW), protected file storge (PFS), and TLS ticketing (TLS). Confidentiality/forward security notions are in rounded boxes, integrity notions in rectangular boxes. Solid lines indicate implications, with numbers referencing the respective theorem in this paper and a plus + when combining several notions. Barred lines denote separations, dotted lines trivial implications, and dashed lines non-tight implications. A star $\star$ or dagger † next to an arrow indicates that the implication holds if puncture invariance (Defs. 5, 8), resp. consistency (Def. 9) is assumed; a $*$ indicates additional assumptions.

The resulting primitive allows authenticated headers and uses tags to enable fine-grained puncturing of ciphertexts. The puncturing tags simplify the exposition of PKW and allow for versatile treatments of the targeted applications: e.g., tags may be chosen via a counter when keeping state or ordering is required, or as random strings when meta-data privacy is a concern (cf. [6]). This contrasts with the foundational work on (non-puncturable) key wrapping [51], where randomness needed for secure wrapping is effectively extracted from the wrapped key in the SIV construction.

We introduce four different security notions for PKW schemes (see Figure 1), three relating to confidentiality (find$-cpa: a classical "real-or-random" notion, find$-rcpa: additionally allowing "real" wrappings, and find$-1cpa: a one-time challenge notion) and one to integrity (of ciphertexts, int-ctxt). They are developed with an eye towards applications, catering to the needs of key hierarchies found in cloud storage systems and the TLS ticketing mechanism. Hence, all four are in a multi-key (or multi-user) setting [5]. The core confidentiality notion, find$-cpa, allows an adversary to obtain, via oracles, "real or random" wrappings of data encryption keys of its choice; the adversary is tasked with deciding which it obtains. The adversary can also puncture master keys at tags of its choice and perform corruptions to obtain current versions of selected master keys (with minimal limitations to prevent trivial wins). The other two confidentiality notions describe a strengthening resp. a relaxation of find$-cpa: find$-rcpa additionally allows the adversary to obtain "real" wrappings of encryption keys via a wrapping oracle, while find$-1cpa allows only one access to the "real or random" wrapping oracle. The integrity notion int-ctxt asks the adversary to create a fresh valid wrapping, given access to wrapping, unwrapping and puncturing oracles. For integrity, no corruption capability is needed in our applications, since they do not aim to uphold integrity guarantees post-corruption; having "multi-entity, forward-secure integrity" for settings where wrapping keys are distributed, e.g., among several servers, may be an interesting avenue for future work, though. For completeness, we also define combined notions for confidentiality and integrity in Appendix C and show that the combination of the separate notions implies it.

To instantiate our new primitive, we provide a simple and generic construction for a PKW scheme based on a PPRF and an AEAD scheme. The core idea is to view the master key as the secret key of a PPRF; wrapping of a selected data encryption key is performed by evaluating the PPRF on the tag to generate a one-time AEAD key, and then using that AEAD key to encrypt the data encryption key. PKW puncturing equates to PPRF puncturing. Depending on the precise assumptions made on the PPRF, we reach our three different levels of confidentiality for the PKW scheme; the integrity notion requires nothing further of the PPRF. In all cases, standard multi-user notions of AEAD security suffice. This straightforward construction is just one example of how a PKW scheme can be implemented. Other

approaches may result in different properties and trade-offs. For example, using a misuse-resistant AEAD scheme [51] could further enable batch puncturing of wrappings under the same tag. Full details of our treatment of PKW can be found in Section 4.

PPRFs. While the precise PPRF security notions we require resemble those in prior work [13, 17, 40, 52], they appear to be, strictly speaking, new. This shows how an application-driven analysis can bring to the surface new requirements on existing primitives. In Section 3 (see also Figure 1), we explore the relations between our different PPRF notions and discuss possible instantiations, e.g., using the GGM construction for PRFs (as adapted to PPRFs in e.g. [40]).

To summarize, we obtain a generic instantiation of PKW, achieving a variety of security notions from standard primitives (AEAD schemes and PRGs).

Application: Forward-secure session resumption. Equipped with our new primitive, we revisit the idea of Aviram, Gellert, and Jager (AGJ) [2] for achieving forward security for the zero round-trip time (0-RTT) data, which can be sent in the pre-shared key mode of TLS 1.3. In this mode, clients rely on a key established in previous sessions (the so called pre-shared key, PSK) to encrypt and send traffic data already together with the very first "hello" message to the server. This reduces the latency of the connection, since the usual TLS handshake happens in parallel with the transmission of this *early data* rather than before it, effectively achieving a 0-RTT setup. However, this speed-up comes at the cost of forward security for the early data.

As observed by AGJ, forward security can be achieved by using puncturing techniques to permanently remove access to the PSKs of completed sessions. Without the PSK, the early data encrypted with it remains confidential even against an attacker who observed the network traffic during the session and compromises the server after session completion. In Section 5, we show how a find$-1cpa-secure PKW scheme can readily be deployed for TLS ticketing to yield forward-secure TLS 1.3 0-RTT resumption that is secure in the sense of a multi-stage key exchange (MSKE) protocol [30]; see also Figure 1. Using PKWs in place of PPRFs (as in AGJ) permits us to take a more generic and abstract viewpoint. This not only directly facilitates constructions offering differing functionality and security guarantees, but also enabled us to identify and correct some technical issues arising in the approach of AGJ.

In particular, building TLS ticketing from PKW allows us to seamlessly switch to a more *privacy-friendly* approach, addressing an open problem in [2]: by sampling tags randomly, we are able to make TLS tickets indistinguishable from random, whereas the AGJ proposal uses a counter in the construction, making their tickets potentially linkable to the time of issuance. Thus our approach can alleviate privacy concerns for TLS ticketing, e.g., regarding tracking users on the web by passively observing network traffic.

The integration of a session resumption protocol into the TLS 1.3 resumption handshake is described in [2, Section 4]. Rephrasing the AGJ proposal in the language of puncturable key wrapping led us to discover conceptual and technical issues in the security model, the proposed protocol, and the proof that prevent the proposal of AGJ from being forward secure, as we discuss in Section 5. Specifically, the security model used in [2] does not reflect the ticketing mechanism of a key exchange protocol in how pre-shared secrets are sampled, registered with parties, and potentially corrupted. Furthermore, the proposed protocol encrypts the TLS resumption master secret RMS in the session ticket. Since RMS is used to derive multiple PSK values, this violates forward security (an adversary learning RMS from one ticket can use it to decrypt prior sessions using a PSK derived from the same RMS). However, this can be easily fixed by ticketing the respective PSK instead of RMS. Finally, we identified overlooked steps and missing underlying assumptions in the AGJ security proof, which were surfaced when applying our PKW formalism. We address all these points in our treatment of forward-secure session resumption for TLS 1.3, see Section 5.

Application: Protected file storage. As a second application example, we show in Section 6 how our new PKW primitive can be used in an encrypted file storage system to give forward security to deleted files. This application is motivated by the current trust assumptions in cloud storage systems, where the confidentiality of the stored data rarely extends to the service provider. Indeed, if the master key in the key hierarchy is managed by the cloud, then the service provider can trivially decrypt any file. The aim of our protected file storage (PFS) system is to provide strong security guarantees for the user, even when encrypted files are outsourced to a malicious or corrupted storage system.

Using a PKW scheme, a client can locally encrypt files under separate data encryption keys, wrap the DEKs with its master key (acting as a KEK) and then outsource both the encrypted files and the

wrapped keys to the cloud. In addition to relieving the user of the need to store anything beyond the master key for the PKW scheme, our PFS system also allows secure *shredding* of files: by puncturing the master key such that a specific wrapped DEK is rendered irrecoverable, the file encrypted by the DEK is made permanently inaccessible, even if the ciphertext is not actually deleted by the cloud storage provider when the client requests it to be. This means that a motivated attacker with access both to the encrypted files *and* the secret key of the user will not be able to compromise the contents of files that were shredded before the user key was compromised. The system hence provides very strong forward security guarantees for shredded files. Crucially in our approach, there is no need for the user to trust the storage provider to actually delete the shredded files, an assumption which would seldom hold in practice due to the presence of backups for disaster recovery purposes (see, e.g., [34]) or bugs in the deletion process [48].

An additional feature of our PFS system is that, in line with current industry practice, it supports key rotation at the KEK level. Key rotation extends the life-time of encrypted data, overcomes usage limits of encryption through rekeying, and supports forward security in practice. It is also important given that the PKW schemes we build have a finite puncturing capability; KEK rotation is then used to restore puncturing capability whenever needed. The multi-key aspect of our PKW security notions readily supports this key rotation.

As core contributions here, we define a syntax for PFS and security notions capturing confidentiality, forward security, and integrity of stored files in a PFS scheme. We show how all of these notions can be achieved by building a PFS scheme from a PKW scheme and an AEAD scheme in a natural and efficient way. We actually provide two different routes to proving our main results on the forward security of PFS, as represented in the first and third column in Figure 1. These routes rely on different security assumptions on the underlying cryptographic components, specifically the PKW scheme used, and result in security theorems with different tightness properties—using a stronger PKW scheme yields a tighter proof of security for the PFS scheme. This in turn relates to the properties required of the underlying PPRF in each of the two routes. While the left, tighter, route requires a PPRF satisfying the strongest security notion (fpr-rro$) as a basic building block—an assumption which, to the best of our knowledge, generally relies on a non-tight (complexity-leveraging) reduction to weaker PPRF notions—it asks less from the building blocks in terms of other properties. Specifically, it avoids the technical requirements of puncture invariance and consistency which we detail in Section 4 and that not all PPRFs may provide, yet which are required for the right, less tight route. The two routes hence show that secure PFS schemes can be constructed from different levels of PKW (and PPRF) schemes; we see this as motivating future work on efficient PKW (or PPRF) constructions that directly fulfill our strong security notions.

We stress that the aim of our PFS system is to showcase how integrating PKWs into existing symmetric key hierarchies can improve security for the *cryptographic core* of secure file storage systems. Building a full-blown system is left to future work.

1.2 Further Related Work

The origins of forward security, in the context of key exchange, date back to Günther [36] and Diffie et al. [24]. A helpful systematization is given by Boyd and Gellert [16].

Green and Miers [35] introduced *puncturable* (public-key) encryption as a means of achieving fine-grained forward security. The ideas of [35] were applied to 0-RTT key exchange and session resumption for TLS 1.3 in [37, 23, 2] as well as symmetric key exchange [3, 15]. The treatment of [15] is for general key exchange, where both parties share a key to a PPRF and puncture it in a semi-synchronized manner. By contrast, our approach to achieving forward security for TLS 1.3 PSK resumption mode using session tickets (in common with [2]) targets the use of puncturable primitives in a "one-sided" setting, where only the server holds the key and performs puncturing operations.

Puncturing techniques have further been used in the context of searchable encryption [56, 55]. Fine-grained forward security is also targeted in Derived Unique Keys Per Transaction (DUKPT) [18]: keys are derived in a tree structure and used in a one-time manner, with the aim of improving security against side-channel attacks on weakly protected devices, e.g., payment terminals.

The idea of *secure outsourced storage* is not new. Blaze [9] designed a "Cryptographic File System" already in 1993 to empower users to encrypt their files, preventing remote file servers used for storage from gaining plaintext access to user data. A rich body of work followed suit, improving on and expanding the security guarantees in the direction of, for example, data integrity and file sharing [47], group collaboration [29], access pattern and metadata hiding [21, 20] and minimizing trust assumptions [45]. There is also a plethora of services running on top of existing storage systems, for example [43, 14]. *Key*

rotation for symmetric encryption is widely used by outsourced storage systems in practice, but was only recently formally treated, see [28] and follow-up works [44, 41] including work using puncturing [54].

Our approach to secure file storage shares the aim of removing the need to trust the storage provider for confidentiality, but we specifically focus on adding *forward security* for individual files. Boneh and Lipton [12] introduced the idea of using key deletion to revoke access to encrypted files, with an emphasis on file backup systems. Their proposal uses linear data structures to store keys, but lacks the fine-grained forward security and key rotation our PFS scheme offers.

A more recent proposal, BurnBox [58], recognizing the difficulty of truly secure file deletion, introduced self-revocable encryption to limit the power of compelled searches of devices. BurnBox achieves fine-grained forward security for deleted files via a tree-based key hierarchy, storing the root in erasable storage. It further hides file metadata in a protected lookup table, an approach we also suggest for our system. On the surface, these properties make BurnBox very similar to our PFS concept. However, the main goal of BurnBox is not forward security, but the much stronger notion of *compelled access security*, which encompasses temporarily revoking file access when device compromise is expected and further goals such as deletion/revocation obliviousness and timing privacy. This forces BurnBox to use highly application-specific approaches, rely on secure storage, and compromise on efficiency (e.g., of file lookups, in favor of privacy). In contrast, our approach is more generic, requires fewer assumptions, and can directly benefit from optimizations of the underlying PKW or PPRF schemes.

2 Preliminaries

We introduce some notation used in this paper and briefly recap syntax and security of (nonce-based) authenticated encryption with associated data [50].

2.1 Notation and Conventions

If a is a string then $|a|$ denotes its length in bits. By $\{0,1\}^n$ we denote the set of all binary strings of length n. By $\{0,1\}^*$ we denote the set of all binary strings of any length, including the empty string, which is denoted ε. The symbol $\|$ denotes concatenation, and we use $a_1\|a_2\|\cdots\|a_n$ as shorthand for the concatenation of strings $a_1, a_2, \ldots, a_n$.

If S is a finite set, we let $x \leftarrow\!\!\$\, S$ denote picking an element of S uniformly at random (u.a.r.) and assigning it to x, and we let $|S|$ denote the size of S. For sets $\mathcal{S}_1, \mathcal{S}_2$, the shorthand $\mathcal{S}_1 \xleftarrow{\cup} \mathcal{S}_2$ denotes $\mathcal{S}_1 \leftarrow \mathcal{S}_1 \cup \mathcal{S}_2$. All sets/spaces associated to cryptographic schemes are assumed non-empty unless otherwise specified.

By $v \leftarrow x$, we mean that the variable v gets assigned the value x. The shorthand $v, w \leftarrow x$ denotes $v \leftarrow x$ and $w \leftarrow x$ for variables v and w. If X is an n-tuple, then by $(x_1, x_2, \ldots, x_n) \leftarrow X$ we denote parsing X into its constituents, which are then individually accessible through variables $x_1, x_2, \ldots, x_n$. If we assign a tuple to a variable through $X \leftarrow (x_1, x_2, \ldots, x_n)$, then we assume an implicit encoding which allows the individual elements to be recovered from X by parsing it into its constituents. We let $|X|$ denote the number of elements in the tuple X. Sometimes we interpret an n-tuple $X = (x_1, x_2, \ldots, x_n)$ as a list and use the shorthand $X \mathrel{+}= x_{n+1}$ to denote adding element x_{n+1} to the list. Formally, this re-assigns to variable X the n+1-tuple $(x_1, \ldots, x_n, x_{n+1})$. Similarly, if $1 \le i \le n = |X|$ then $X \mathrel{-}= x_i$ means that the ith element is removed from the list. If x_i is not in the list, the result of $X \mathrel{-}= x_i$ is the original, unchanged list X. By $X \leftarrow ()$ we mean that variable X is initialized to an empty list (0-tuple). If $X = (x_1, x_2, \ldots, x_n)$ and $Y = (y_1, y_2, \ldots, y_m)$ are two tuples of length n and m respectively, then $X\|Y$ denotes the $n+m$-tuple $(x_1, \ldots, x_n, y_1, \ldots, y_m)$.

By $\wedge$ we denote the logical AND operator, and by $\vee$ inclusive OR. We use $\perp$ (bot) as a special symbol to denote rejection, and it is assumed to not be in $\{0,1\}^*$. Both inputs and outputs to algorithms can be $\perp$. We adopt the convention that if any input to an algorithm is $\perp$, then its output is $\perp$ as well. When specifying syntax, we sometimes write $y/\perp \leftarrow A(\cdot)$ to explicitly show that the output of algorithm A is a string y or $\perp$. Algorithms may be randomized unless otherwise indicated.

If A is an algorithm, we let $y \leftarrow A^{\mathrm{O}_1,\cdots}(x_1, \ldots; r)$ denote running A on inputs $x_1, \ldots$ and coins r, with oracle access to $\mathrm{O}_1, \ldots$, and assigning the output to y. We use the shorthand notation $y \leftarrow\!\!\$\, A^{\mathrm{O}_1,\cdots}(x_1, \ldots)$ to denote picking r at random and letting $y \leftarrow A^{\mathrm{O}_1,\cdots}(x_1, \ldots; r)$. An adversary is considered an algorithm. All algorithms implicitly perform formatting checks on any input they receive and halt and return $\perp$ if the formatting is incorrect.

We use the game-playing framework of [7]. By $\Pr[\,\mathbf{G}(\mathcal{A}) \Rightarrow x\,]$ we denote the probability that the execution of game $\mathbf{G}$ with adversary $\mathcal{A}$ results in the game outcome taking value x. By true and false

we denote the boolean values of true and false. We identify true with the value 1 and false with the value 0. A $\neg$ preceding a boolean variable denotes negation. The expression $x = y$ is a boolean which evaluates to true if x is equal to y, false otherwise. Sometimes the shorthand notation $\Pr[\mathbf{G}(\mathcal{A})]$ will be used as an abbreviation for $\Pr[\mathbf{G}(\mathcal{A}) \Rightarrow \text{true}]$. In games, integer variables, strings, set variables and boolean variables (such as the win flag) are assumed initialized, respectively, to 0, the empty string ε, the empty set $\emptyset$, and false, unless otherwise specified.

All of our security notions are given in a multi-user version [5]. In the games defining the notions, this is captured by an oracle NEW which the adversary can call to initialize a new key. We use a counter u to keep track of the number of initialized keys. All other oracles take as input an index i that indicates under which key the query should be executed. We assume that the adversary only asks queries with index $0 < i \leq u$, otherwise the query is aborted with $\perp$ as response.

2.2 AEAD

We recall the syntax and security of a nonce-based authenticated encryption with associated data (AEAD) scheme as defined by Rogaway [50].

Definition 1 (AEAD scheme)**.** An *authenticated encryption with associated data* scheme, $\mathsf{AEAD} = (\mathsf{Enc}, \mathsf{Dec})$, is a pair of algorithms with four associated sets; the secret-key space $\mathcal{SK}$, the nonce space $\mathcal{N}$, the associated data space $\mathcal{AD}$ and the message space $\mathcal{M}$. Further associated with AEAD is a ciphertext-length function $\mathsf{cl}\colon \mathbb{N} \to \mathbb{N}$. The algorithms of AEAD operate as follows.

- Via $C \leftarrow \mathsf{Enc}(sk, N, ad, M)$, the deterministic encryption algorithm Enc on input the secret key $sk \in \mathcal{SK}$, a nonce $N \in \mathcal{N}$, associated data $ad \in \mathcal{AD}$ and a message $M \in \mathcal{M}$ produces a ciphertext $C \in \{0,1\}^{\mathsf{cl}(|M|)}$.
- Via $M/\perp \leftarrow \mathsf{Dec}(sk, N, ad, C)$, the deterministic decryption algorithm Dec on input the secret key $sk \in \mathcal{SK}$, a nonce $N \in \mathcal{N}$, associated data $ad \in \mathcal{AD}$ and a ciphertext $C \in \{0,1\}^*$ produces a message $M \in \mathcal{M}$ or, to indicate failure, the special symbol $\perp$.

Correctness of a nonce-based AEAD scheme stipulates that $\mathsf{Dec}(sk, N, ad, \mathsf{Enc}(sk, N, ad, M)) = M$ for all $sk \in \mathcal{SK}$, $N \in \mathcal{N}$, $ad \in \mathcal{AD}$ and $M \in \mathcal{M}$.

For security, we consider confidentiality (ind$-cpa) as well as ciphertext integrity (int-ctxt).

Definition 2 (AEAD confidentiality, ind$-cpa)**.** Let AEAD be a nonce-based AEAD scheme and let game $\mathbf{G}^{\text{ind\$-cpa}}_{\mathsf{AEAD}}$ be defined as in Figure 2. We define the *confidentiality (*ind$-cpa*)* advantage of an adversary $\mathcal{A}$ against AEAD as

$$\mathbf{Adv}^{\text{ind\$-cpa}}_{\mathsf{AEAD}}(\mathcal{A}) = 2\left|\Pr\left[\mathbf{G}^{\text{ind\$-cpa}}_{\mathsf{AEAD}}(\mathcal{A}) \Rightarrow \text{true}\right] - \frac{1}{2}\right|.$$

Definition 3 (AEAD integrity, int-ctxt)**.** Let AEAD be a nonce-based AEAD scheme and let game $\mathbf{G}^{\text{int-ctxt}}_{\mathsf{AEAD}}$ be defined as in Figure 2. We define the *integrity (*int-ctxt*)* advantage of an adversary $\mathcal{A}$ against AEAD as

$$\mathbf{Adv}^{\text{int-ctxt}}_{\mathsf{AEAD}}(\mathcal{A}) = \Pr\left[\mathbf{G}^{\text{int-ctxt}}_{\mathsf{AEAD}}(\mathcal{A}) \Rightarrow \text{true}\right].$$

3 Puncturable PRFs

Puncturable PRFs (PPRFs) were conceived of independently in [13], [17] and [40]. We recall the definition from Sahai and Waters [52], but restrict our attention to PPRFs with deterministic puncturing algorithms.

Definition 4 (PPRF)**.** A *puncturable pseudorandom function* $\mathsf{PPRF} = (\mathsf{KeyGen}, \mathsf{Eval}, \mathsf{Punc})$ is a triple of algorithms with three associated sets; the secret-key space $\mathcal{SK}$, the domain $\mathcal{X}$ and the range $\mathcal{Y}$.

- Via $sk \leftarrow\!\!\$\ \mathsf{KeyGen}()$, the probabilistic key generation algorithm KeyGen, taking no input, outputs the secret key $sk \in \mathcal{SK}$.
- Via $y/\perp \leftarrow \mathsf{Eval}(sk, x)$, the function evaluation algorithm Eval, on input the secret key sk and an element $x \in \mathcal{X}$ outputs $y \in \mathcal{Y}$ or, to indicate failure, $\perp$.

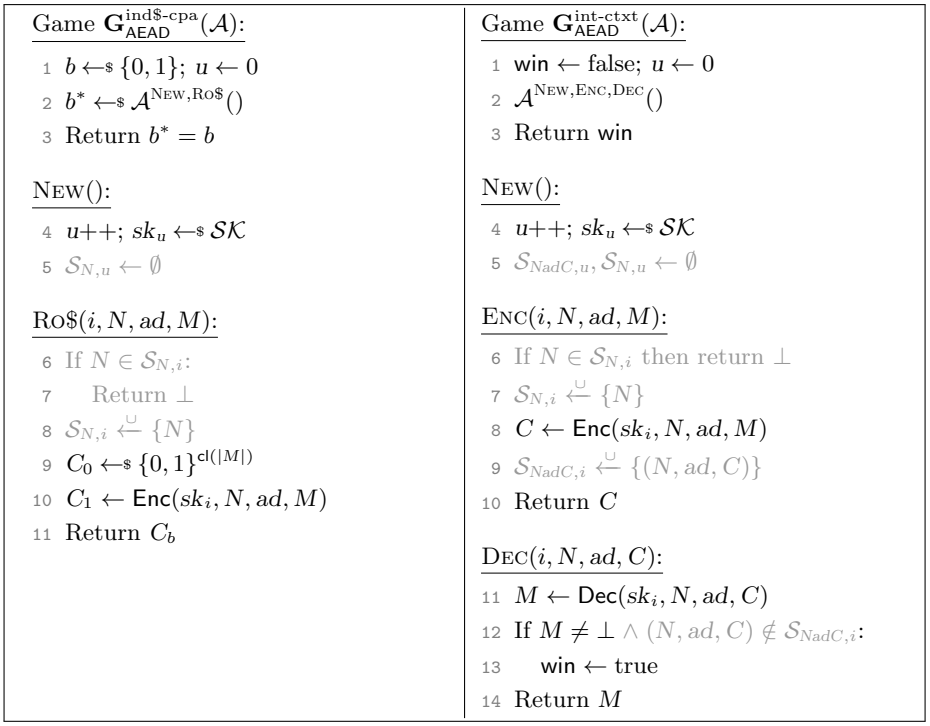

Figure 2: Games formalizing confidentiality (ind\$-cpa) and integrity of ciphertexts (int-ctxt) of an authenticated encryption scheme with associated data AEAD. Grey code prevents trivial attacks and ensures that the adversary is nonce-respecting.

- Via $sk' \leftarrow \mathsf{Punc}(sk, x)$, the deterministic puncturing algorithm Punc, on input the secret key sk and an element $x \in \mathcal{X}$ outputs an updated secret key $sk' \in \mathcal{SK}$.

For *correctness* we require that for all $sk \in \mathcal{SK}$ and all $x, y \in \mathcal{X}$:

(1) $\Pr[\,\mathsf{Eval}(sk_0, x) \neq \perp \mid sk_0 \leftarrow\$ \mathsf{KeyGen}()\,] = 1$.

(2) If $sk' \leftarrow \mathsf{Punc}(sk, x)$ and $y \neq x$, then $\mathsf{Eval}(sk, y) = \mathsf{Eval}(sk', y)$.

(3) If $sk' \leftarrow \mathsf{Punc}(sk, x)$, then $\mathsf{Eval}(sk', x) = \perp$.

Requirement (1) ensures that for any freshly generated secret key sk_0 and for any $x \in \mathcal{X}$, $\mathsf{Eval}(sk_0, x)$ will not be $\perp$. Requirement (2) says that puncturing any secret key sk on x only affects the evaluation of x. Requirement (3) demands that the evaluation of a punctured point will always be $\perp$.

Note that, as a consequence of requirement (3), our definition excludes PPRFs which output an element in the range $\mathcal{Y}$ when evaluated on a punctured point, such as e.g. the privacy-preserving PPRFs of Boneh et al. [10, 11]. Why, then, do we impose this strict requirement?

While we must clearly demand that the evaluation under the unpunctured secret key of a point x is different from the evaluation of x under a key which has been punctured on x (for security), the latter must not necessarily be $\perp$. Indeed, in prior work [2, 52] this has not been an requirement in the definition of PPRFs (neither explicitly in correctness, nor implicitly by security). However, as we show in Section 5, these weaker definitions do not suffice to provide integrity in applications which use a PPRF for key derivation. In fact, even the stronger of the two PPRF security notions presented by Aviram, Gellert and Jager in [2] is too weak and permits a concrete attack on the integrity of the session resumption protocol which they design for TLS. Hence, to prevent such attacks on our constructions, we choose to restrict ourselves to PPRFs which output $\perp$ after puncturing, and argue that this captures the intuition that function evaluation on punctured points should "fail". Since our applications do not need the PPRF to hide the points on which it was punctured, exclusion of the powerful but inefficient privacy-preserving PPRFs of [11, 10, 19] is a warranted compromise.

Following [2], we define an additional property of PPRFs called "puncture invariance" which demands that the scheme is insensitive to the order in which punctures are performed. I.e., the puncturing operation is commutative with respect to the resulting secret key. As noted in [2], this property enables

reductions that change the order of punctures without an adversary later compromising the secret key noticing; this is necessary for example to have our single-challenge notion (fpr-1ro$) imply our core PPRF notion (fpr-ro$), as we shall see.

Definition 5 (PPRF puncture invariance)**.** A puncturable pseudorandom function $\mathsf{PPRF} = (\mathsf{KeyGen}, \mathsf{Eval}, \mathsf{Punc})$ is *puncture invariant* if for all keys $sk \in \mathcal{SK}$ and all $x_0, x_1 \in \mathcal{X}$ it holds that

$$\mathsf{Punc}(\mathsf{Punc}(sk, x_0), x_1) = \mathsf{Punc}(\mathsf{Punc}(sk, x_1), x_0).$$

3.1 PPRF Security and Relations

We define three security notions for PPRFs, all in the multi-user setting [5], capturing the combined forward security and pseudorandomness goals, or *forward pseudorandomness* (fpr) for short. Let us start with our core *forward pseudorandomness* notion (fpr-ro$), given in Figure 3. It is an extension of classical PRF security, where the adversary is given oracle access (Ro$-Eval) either to the real function evaluated on a hidden key, or a lazily-sampled random function. Forward security is captured through access to a puncturing oracle (Punc) as well as corruption oracle (Corr), through which the adversary can obtain secret keys that have been punctured on all challenge points.

Our second, stronger notion, *forward pseudorandomness with real evaluations* (fpr-rro$), in addition gives the adversary access to a real evaluation oracle (Eval), capturing that real evaluations do not help distinguishing challenge outputs (even post-corruption).

In our third, weaker notion, *single-challenge forward pseudorandomness* (fpr-1ro$), the adversary only gets a single challenge evaluation under each key. The challenge is obtained from oracle New-Ro$-Eval, which on input a domain point x returns either the real function evaluation of x under the (unpunctured) secret key (in the "real" world), or a string drawn u.a.r. from $\mathcal{Y}$ (in the "ideal" world). Additionally the adversary obtains the secret key punctured on x. As usual, the adversary wins if it can distinguish the real world from the ideal one.

Definition 6 (PPRF security (fpr-ro$, fpr-rro$, fpr-1ro$))**.** Let PPRF be a puncturable pseudorandom function. We define the advantage of an adversary $\mathcal{A}$ against the forward pseudorandomness $X \in \{\text{fpr-ro\$}, \text{fpr-rro\$}, \text{fpr-1ro\$}\}$ of PPRF as

$$\mathbf{Adv}^{X}_{\mathsf{PPRF}}(\mathcal{A}) = 2 \left| \Pr\left[\mathbf{G}^{X}_{\mathsf{PPRF}}(\mathcal{A}) \Rightarrow \text{true} \right] - \frac{1}{2} \right|,$$

where game $\mathbf{G}^{X}_{\mathsf{PPRF}}(\mathcal{A})$ is given in Figure 3.

Comparison to prior work. Our PPRF notions resemble those in prior work, but also differ in several ways. For example, fpr-1ro$ is similar to the non-adaptive notion in [52, 2], but restricted to a single challenge. Through a multi-key hybrid argument [5], their notion implies ours. The adaptive "*rand*" notion of [2] most closely corresponds to our fpr-rro$ notion, but our notion provides the adversary with more flexibility by both allowing multiple real-or-random challenge evaluations under each key (compared to a single evaluation under the single key in [2]) and giving it access both to a separate puncturing oracle (the *rand* experiment only punctures on the single challenge point) and corruption oracle, thereby allowing multiple key compromises of keys punctured on points chosen by the adversary. Our middle notion fpr-ro$ is, to the best of our knowledge, new.

PPRF relations. Figure 1 (on page 4) shows the relations between our PPRF security notions. The trivial implications (dotted lines) immediately arise from restricting the adversary. As an example, fpr-rro$ implies fpr-ro$ because an adversary against the fpr-rro$ security can simply ignore the Eval-oracle. Similarly fpr-ro$ implies fpr-1ro$.

In the other direction, fpr-1ro$ implies fpr-ro$ for any puncture-invariant PPRF PPRF. That is, for any adversary $\mathcal{A}$ against the fpr-ro$ security of PPRF, there exists an adversary $\mathcal{B}$ running in approximately the same time as $\mathcal{A}$ such that $\mathbf{Adv}^{\text{fpr-ro\$}}_{\mathsf{PPRF}}(\mathcal{A}) \leq q_{ro\$} \cdot \mathbf{Adv}^{\text{fpr-1ro\$}}_{\mathsf{PPRF}}(\mathcal{B})$, via a standard hybrid argument, where puncture invariance ensures that reorderings of punctures do not affect simulation of the later-corrupted secret key. The theorem statement and more details of the proof are in Appendix A.

Via a non-tight reduction, we can also show that fpr-ro$ implies fpr-rro$ for a puncture-invariant PPRF. This is again via a hybrid argument, which however now involves guessing the input to the challenge query Ro$-Eval under each key (so-called complexity leveraging [13, 17]), resulting in reduction

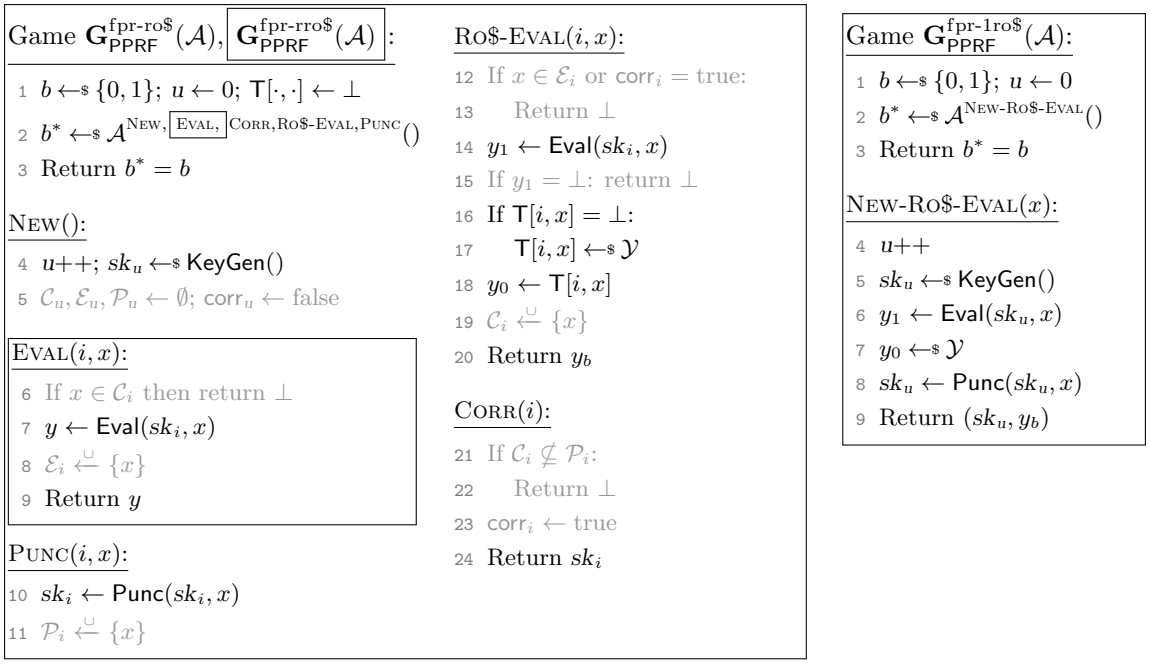

Figure 3: Left: Games defining real-or-$ (fpr-ro$, without the boxed Eval oracle) and real-and-real-or-$ (fpr-rro$, with $\mathcal{A}$ having access to Eval) forward pseudorandomness. Right: Game defining one-time forward pseudorandomness (fpr-1ro$) PPRF security. Grey code prevents trivial attacks.

loss proportional to the size of the PPRF domain. As above, the proof starts with a hybrid argument from at most $q_{ro\$}$ queries to oracle Ro$-Eval per key to a single. The resulting game can be simulated by a fpr-ro$ adversary which guesses the input to the Ro$-Eval query under each key such that it can get the challenge response in advance, and then puncture and corrupt the key. The compromised key is used to simulate queries to oracle Eval. Because of the guessing step, the loss of the reduction is proportional to the size of the domain. The details are in Appendix A.

Instantiations from the literature. One, by now folklore, way of building a PPRF is to use the GGM PRF construction via a tree of pseudorandom-generator (PRG) evaluations [33], extended with a puncturing algorithm, as first noted by [13, 17, 40]. The core idea to enable puncturing on a domain point x in a GGM PRF is to update the secret key, removing nodes on the path to x in the PRG tree and adding all nodes on the co-path from the root to x. For a more in-depth description and argument of security we refer to [2, 40]. Note that the GGM-based construction is correct and puncture invariant, and hence, via our established relations, yields an fpr-ro$-secure PPRF. For small domains where complexity leveraging is acceptable, it further achieves the stronger fpr-rro$ notion. Additionally, for this specific construction, adaptive security can be achieved with a loss factor that is only quasi-polynomial in the input length, improving greatly over the exponential loss of complexity leveraging [32]. An alternative construction for a PPRF with security based on the Strong RSA assumption can be found in [2].

4 Puncturable Key Wrapping

We now present our core cryptographic primitive, *puncturable key wrapping* (PKW). With puncturable key wrapping, we merge the notion of key wrapping, originally extensively studied by Rogaway and Shrimpton [51], with tag-based puncturable encryption [35], adapted to the symmetric setting, to capture forward security through puncturing. Puncturable key wrapping, beyond the key K to be wrapped, hence takes a *tag* T used as a pointer for puncturing, as well as optional associated *header* data H which is authenticated along with the wrapped key (akin to associated data in AEAD). In the following, we give syntax, security, and further notions for this new primitive.

Definition 7 (PKW scheme)**.** A *puncturable key-wrapping scheme* $\mathsf{PKW} = (\mathsf{KeyGen}, \mathsf{Wrap}, \mathsf{Unwrap}, \mathsf{Punc})$ is a 4-tuple of algorithms with four associated sets; the secret-key space $\mathcal{SK}$, the tag space $\mathcal{T}$, the header space $\mathcal{H}$ and the wrap-key space $\mathcal{K}$. Associated to the scheme is a ciphertext-length function $\mathsf{cl} : \mathbb{N} \to \mathbb{N}$. The algorithms of PKW operate as follows.

- Via $sk \leftarrow\!\!\$\ \mathsf{KeyGen}()$, the probabilistic key generation algorithm KeyGen, taking no input, outputs a secret key $sk \in \mathcal{SK}$.
- Via $C/\perp \leftarrow \mathsf{Wrap}(sk, T, H, K)$, the deterministic wrapping algorithm Wrap on input a secret key $sk \in \mathcal{SK}$, a tag $T \in \mathcal{T}$, a header $H \in \mathcal{H}$ and a key $K \in \mathcal{K}$ outputs a ciphertext $C \in \{0,1\}^{\mathsf{cl}(|K|)}$ or, to indicate failure, $\perp$.
- Via $K/\perp \leftarrow \mathsf{Unwrap}(sk, T, H, C)$, the deterministic unwrapping algorithm Unwrap on input a secret key $sk \in \mathcal{SK}$, a tag $T \in \mathcal{T}$, a header $H \in \mathcal{H}$ and a ciphertext $C \in \{0,1\}^*$ returns a key $K \in \mathcal{K}$ or, to indicate failure, $\perp$.
- Via $sk' \leftarrow \mathsf{Punc}(sk, T)$, the deterministic puncturing algorithm Punc on input a secret key $sk \in \mathcal{SK}$ and a tag $T \in \mathcal{T}$ returns a potentially updated secret key $sk' \in \mathcal{SK}$.

Correctness of a PKW scheme intuitively demands that a wrapped key can be recovered from its wrapping ciphertext unless the secret key has been punctured on the tag used for the wrapping step, i.e., even if the secret key has been punctured on other tags. Formally, we require that for all $T \in \mathcal{T}$, $H \in \mathcal{H}$, $K \in \mathcal{K}$, and all tuples $\bar{T}_1, \bar{T}_2 \in \mathcal{T}^*$ where $T \notin \bar{T}_1$ and $T \notin \bar{T}_2$,

$$\Pr\left[\, \mathsf{Unwrap}(sk_{\setminus \bar{T}_1}, T, H, \mathsf{Wrap}(sk_{\setminus \bar{T}_2}, T, H, K)) = K \mid sk \leftarrow\!\!\$\ \mathsf{KeyGen}() \,\right] = 1.$$

Here $sk_{\setminus (T_1, T_2, \ldots, T_n)} = \mathsf{Punc}(\ldots(\mathsf{Punc}(\mathsf{Punc}(sk, T_1), T_2), \ldots), T_n)$ is shorthand for the secret key obtained via puncturing sk in order on $T_1, \ldots, T_n \in \mathcal{T}$.

Analogously to Definition 5 for PPRFs, we also define *puncture invariance* for PKW schemes, demanding that the order of punctures does not affect the resulting secret key.

Definition 8 (PKW puncture invariance)**.** A puncturable key-wrapping scheme $\mathsf{PKW} = (\mathsf{KeyGen}, \mathsf{Wrap}, \mathsf{Unwrap}, \mathsf{Punc})$ is *puncture invariant* if for all keys $sk \in \mathcal{SK}$ and all tags $T_0, T_1 \in \mathcal{T}$ it holds that

$$\mathsf{Punc}(\mathsf{Punc}(sk, T_0), T_1) = \mathsf{Punc}(\mathsf{Punc}(sk, T_1), T_0).$$

That is, the current state of the secret key only depends on the set of tags on which it has been punctured, and not on the order in which the punctures were performed.

Additionally, we introduce a property of PKW schemes which we call *consistency*, inspired by the definition of consistent puncturable signature schemes in [8]. A consistent PKW scheme is one for which the output of algorithm Wrap only depends on the tag, header and wrap-key input, and not on the (puncturing) state of the secret key—except for when the output is $\perp$ due to puncturing.

Definition 9 (PKW consistency)**.** A puncturable key wrapping scheme $\mathsf{PKW} = (\mathsf{KeyGen}, \mathsf{Wrap}, \mathsf{Unwrap}, \mathsf{Punc})$ is *consistent* if for all keys $K \in \mathcal{K}$, all headers $H \in \mathcal{H}$, all tags $(T_1, \ldots, T_n) \in \mathcal{T}^*$ and all $T \in \mathcal{T} \setminus \{T_1, \ldots, T_n\}$ it holds that

$$\Pr\left[\, \mathsf{Wrap}(sk, T, H, K) = \mathsf{Wrap}(sk_{\setminus (T_1, \ldots, T_n)}, T, H, K) \mid sk \leftarrow\!\!\$\ \mathsf{KeyGen}() \,\right] = 1.$$

Puncture invariance and consistency guarantee a kind of indifference of the PKW scheme with respect to puncturing, allowing sequences of punctures and wrappings to be flexibly reordered without affecting the scheme's future behavior. As we shall see, these properties are important to consider when deploying PKW schemes in, and proving the security of, higher-level applications.

4.1 PKW Security

Confidentiality. Following Rogaway and Shrimpton [51], we adopt indistinguishability from random bits (ind$) as the appropriate notion to model confidentiality for (puncturable) key-wrapping schemes. Our three confidentiality notions, formalized in Figure 5, capture forward security in the sense that the confidentiality guarantees hold also after compromise of the secret key, given that it has been appropriately punctured prior to corruption to avoid trivial wins. As before, they are all in the multi-key (or multi-user) setting [5]. To focus on forward security, we separate confidentiality (with forward security) and integrity (below) into distinct notions, contrasting with the combined notion in [51]. We give a combined notion in Appendix C, also capturing CCA-style active attacks, and show that it is equivalent to the junction of our separate notions.

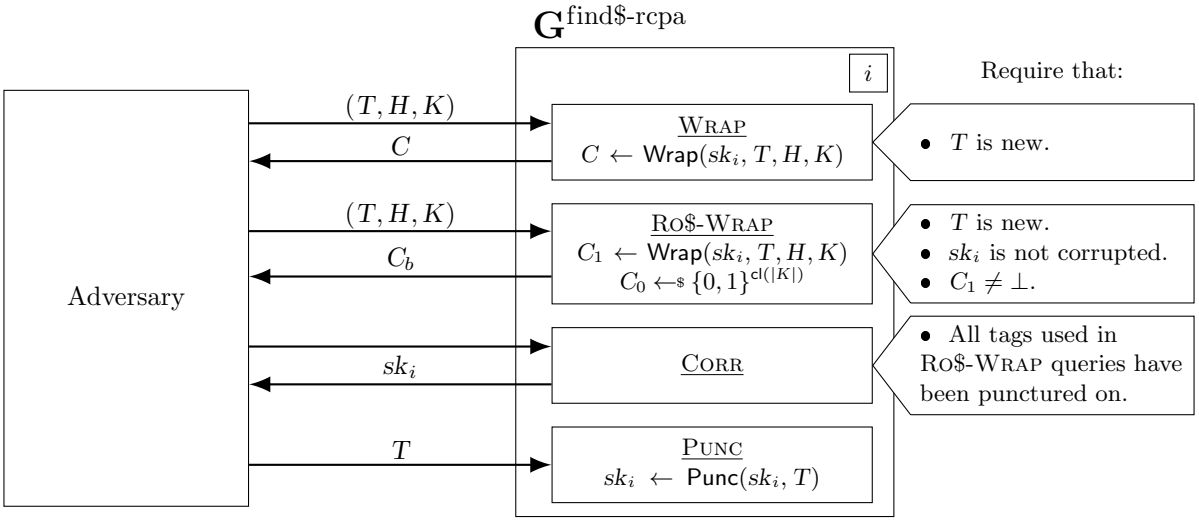

Figure 4: Illustration of the adversary's interactions in the PKW find$-rcpa security game, focusing on one key sk_i. (Oracle NEW generates a new key sk_i and gives the adversary the ability to query the other oracles under key sk_i.) If oracle WRAP is omitted, the illustration for find$-cpa security is obtained.

Our first confidentiality notion, which we call find$-cpa, can be viewed as a form of ind$-cpa security adapted to the PKW setting. The adversary is given access to a challenge wrapping oracle RO$-WRAP, which on input a key index i, a tag T, a header H and a key K chosen by the adversary, returns either an honest wrapping of K under secret key sk_i, or a random bit-string of length $\mathsf{cl}(|K|)$. Forward security is captured via a corruption oracle CORR which allows the adversary to compromise the current version of a secret key sk_i, given that all tags used in challenge queries under sk_i must be punctured on at the time of corruption (via the puncturing oracle PUNC). Focusing on fine-grained forward security, we restrict the adversary to use tags only *once* for wrapping and call this behavior *tag-respecting* (akin to a nonce-respecting adversary in authenticated encryption); this enables puncturing of *individual* ciphertexts.[2]

Guided by the envisioned usage of puncturable key-wrapping schemes, our second, stronger confidentiality notion, find$-rcpa, additionally gives the adversary access to real wrappings that it does not have to puncture on via an additional oracle WRAP. The rationale behind the notion is that although find$-cpa provides forward security for all wrapped keys which have been punctured on at the time of compromise, it does not capture the potential leakage from unpunctured ciphertexts which the adversary gains insight into by corrupting. That is, we would like to ensure that there is a form of independence across key wrappings produced with distinct tags. This is motivated by what we believe to be realistic attack scenarios for applications which use a PKW scheme for key management—such as our protected file storage system (to be defined in Section 6). In such a system, normal usage implies the existence of some unpunctured ciphertexts (corresponding to non-shredded files) at any given time, and hence in particular at the time of a key compromise. The idea of find$-rcpa security is that compromising ciphertexts generated with tags that have not been punctured on, should not give the adversary a higher advantage in distinguishing challenge ciphertexts from random bits. Figure 4 shows an illustration of the game and the restrictions on the oracles to prevent trivial attacks.

Lastly, we also introduce a one-time security notion, find$-1cpa, which only provides the adversary with one challenge output and the punctured secret key, per key. As we will see, together with puncture invariance and consistency, find$-1cpa turns out to be sufficiently strong to achieve full security in the applications we are interested in.

Definition 10 (PKW confidentiality (find$-cpa, find$-rcpa, find$-1cpa))**.** Let PKW be a puncturable key-wrapping scheme. We define the advantage of an adversary $\mathcal{A}$ against the forward indistinguishability $X \in \{\text{find\$-cpa}, \text{find\$-rcpa}, \text{find\$-1cpa}\}$ of PKW as

$$\mathbf{Adv}^{X}_{\mathsf{PKW}}(\mathcal{A}) = 2\left|\Pr\left[\,\mathbf{G}^{X}_{\mathsf{PKW}}(\mathcal{A}) \Rightarrow \text{true}\,\right] - \frac{1}{2}\right|,$$

[2]We note that a stronger formalization is possible where tag reuse is allowed: by storing and checking the whole tuple (T, H, K) in the sets $\mathcal{S}_{T,i}$ instead of only T, one can demand wraps to look random except when this is impossible due to entirely repeating inputs. This could cater to applications interested in "batch puncturing" [35], i.e., revoking access to multiple wrapped keys via a single puncturing call. Such stronger notions would also require stronger building blocks, as we will see below.

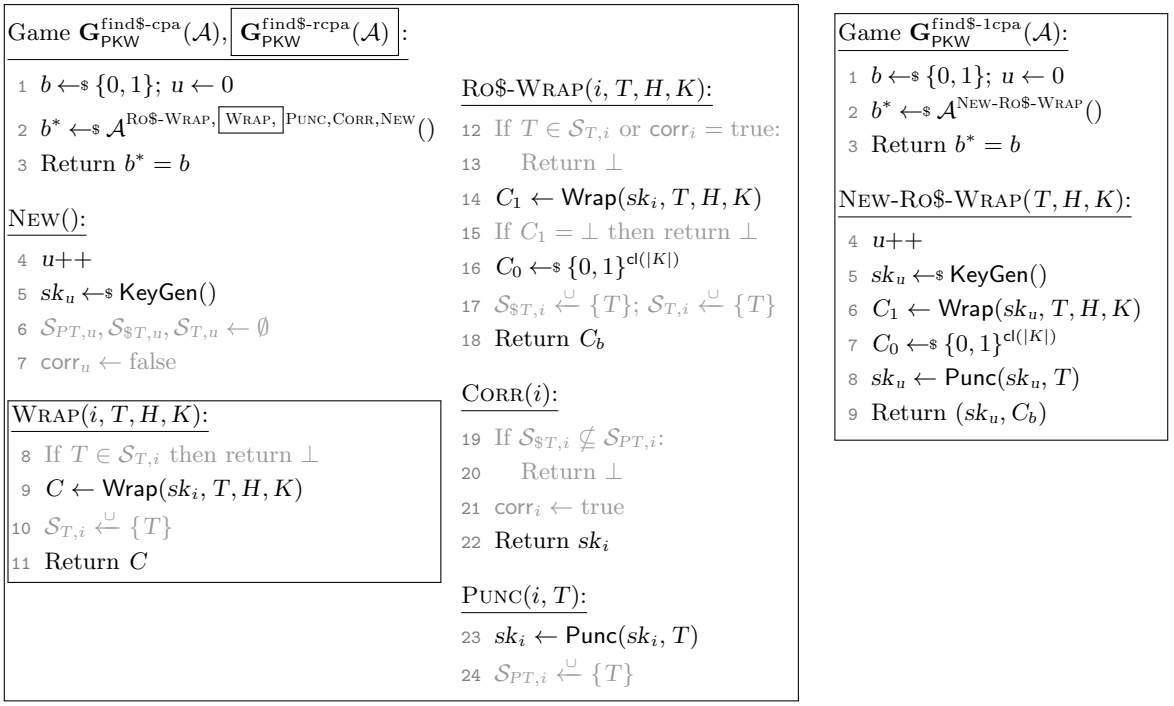

Figure 5: Left and middle: Forward security and privacy find$-cpa (without access to the boxed Wrap oracle) / find$-rcpa (with access to Wrap) of a puncturable key-wrapping scheme PKW. Right: One-time privacy and forward security find$-1cpa security of a puncturable key-wrapping scheme PKW. Grey code prevents trivial attacks and ensures that unique tags are used for wrapping.

where $\mathbf{G}^{X}_{\mathsf{PKW}}(\mathcal{A})$ is defined in Figure 5.

Integrity. In addition to the confidentiality notions we also define (multi-key) *integrity of ciphertexts* (int-ctxt) for PKW schemes as shown in Figure 6. Here, the adversary is given oracle access to wrapping (Wrap), unwrapping (Unwrap), and puncturing (Punc). Its goal is to forge a ciphertext (together with a tag and a header) that was not output by Wrap, or for which the tag was punctured on via Punc, and that unwraps to something other than the error symbol $\perp$. Note that we particularly treat ciphertexts under punctured tags as valid forgery attempts, even if previously output by Wrap. This ensures that after puncturing on a tag, no ciphertext with that tag will be accepted any more, which is sometimes referred to as *replay protection*. As for the confidentiality notions, we require that the adversary is tag-respecting.

Definition 11 (PKW integrity (int-ctxt))**.** Let PKW be a puncturable key-wrapping scheme. We define the advantage of an adversary $\mathcal{A}$ against the *integrity of ciphertexts* of PKW as

$$\mathbf{Adv}^{\text{int-ctxt}}_{\mathsf{PKW}}(\mathcal{A}) = \Pr\left[\mathbf{G}^{\text{int-ctxt}}_{\mathsf{PKW}}(\mathcal{A}) \Rightarrow \text{true}\right],$$

where $\mathbf{G}^{\text{int-ctxt}}_{\mathsf{PKW}}(\mathcal{A})$ is defined in Figure 6.

Notably, in the integrity setting, forging a valid ciphertext becomes trivial if one would allow the adversary to compromise the secret key. Forward security hence seems to only make sense in scenarios where *two copies* of the key are available simultaneously, one "more punctured" than the other. The challenge then would be to forge a ciphertext on a punctured tag T using access to the compromised, more punctured key, such that the ciphertext unwraps under the less punctured key (which has not been punctured on T). This could be interesting, e.g., in a setting where punctured keys are distributed across servers. We leave extending puncturing to the distributed setting as future work, but consider the amalgamation of (forward-secure) confidentiality and integrity in Appendix C.

Relations between PKW notions. We briefly explain how the PKW confidentiality notions are related. See Figure 1 for an overview of all security notions and their relations. Beginning from strong to weak: the trivial implications (dotted arrows) arise directly from restricting the adversary. As an

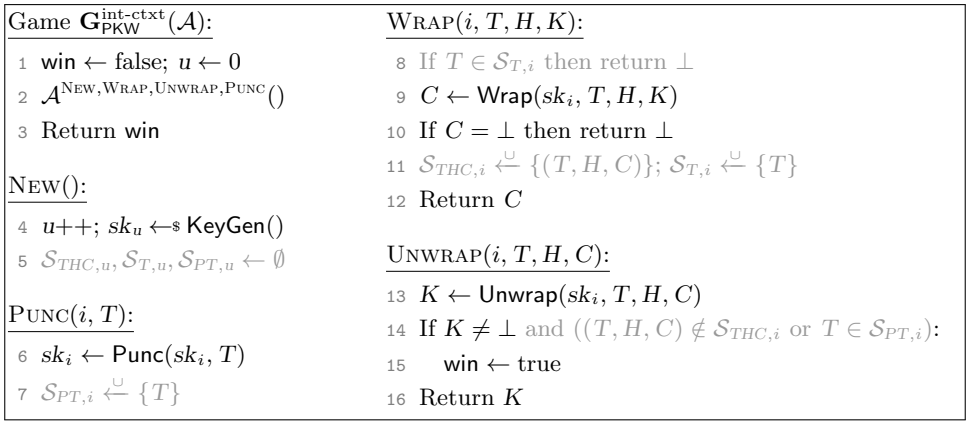

Figure 6: Integrity of ciphertexts of a puncturable key-wrapping scheme PKW. Grey code prevents trivial attacks and ensures that tags are not repeated in wrap queries.

example, find$-rcpa implies find$-cpa because an adversary against the find$-rcpa security can simply ignore the Wrap-oracle.

In the opposite direction the relations are more complex. Generally, find$-1cpa does not imply find$-cpa. Showing the separation is straightforward: Modify any find$-1cpa secure scheme so that Wrap outputs a fixed string when receiving an already-punctured tag as input. This makes challenge wraps on punctured tags—which are available in the find$-cpa game, but not in find$-1cpa—easily distinguishable. In contrast, for the special case of a PKW scheme that is puncture invariant and consistent, and additionally for which attempting to wrap using a punctured tag always results in $\perp$ (i.e., $\mathsf{Wrap}(sk_{\setminus \bar{T}}, T, \cdot, \cdot) = \perp$ if $T \in \bar{T} \subseteq \mathsf{PKW}.\mathcal{T}$)[3], find$-1cpa implies find$-cpa via a hybrid argument. We formally prove both relations in Appendix B.

Lastly, assuming a (forward) secure source of pseudorandomness, such as a fpr-ro$ secure PPRF, find$-rcpa is strictly stronger than find$-cpa. The separation relies on the fact that in the find$-cpa game, an adversary must puncture on all tags which have been used for wrapping before compromising the secret key; a restriction which is not imposed on tags queried to oracle Wrap in the find$-rcpa game. This can be used to construct a scheme which leaks a copy of the original, unpunctured secret key when punctured *only once* on a hidden, special tag $\hat{T}$, which can only be learned by wrapping under a different, fixed and publicly known tag T_0. Tag $\hat{T}$ is accessible to an adversary in the find$-rcpa game via oracle Wrap, but not to a find$-cpa adversary. The latter can learn $\hat{T}$ only through a Ro$-Wrap call on T_0, forcing it to also puncture on T_0 and thereby destroying the key copy. We give the details in Theorem 14, Appendix B.

4.2 Instantiating PKW from PPRF and AEAD

Next, we give a generic construction of a PKW scheme, formalized in Figure 7. The construction uses an authenticated encryption scheme with associated data AEAD to encrypt (wrap) keys, using a new AEAD key together with a fixed nonce N_0 for each key-wrap. The keys of AEAD are generated by a pseudorandom function PPRF on input the wrap tag, the key of which is the secret key of the PKW scheme. This allows AEAD keys to be "forgotten" via puncturing the PPRF key, thereby rendering the key-wrap ciphertexts unrecoverable. The construction is inspired by, and re-captures, the generic construction of a "0-RTT session resumption protocol" by Aviram, Gellert, and Jager [2], with the difference that we use a *nonce-based* AEAD scheme, following practically deployed schemes like AES-GCM or ChaCha20-Poly1305, rather than a probabilistic one.

The only technical requirement for our construction is that the range of PPRF matches the key space of AEAD. The key space of the resulting PKW scheme is the key space of PPRF, the tag space the PPRF domain, the header space the associated data space of AEAD, and the wrap-key space the message space of AEAD. The ciphertext-length function cl for PKW is that of AEAD.

The following sequence of results shows that, given certain properties of the underlying PPRF and AEAD schemes, our construction PKW[PPRF, AEAD] achieves puncture invariance and consistency (Lemma 1), different levels of forward indistinguishability depending on the underlying PPRF strength (Theorems 2–4), as well as integrity of ciphertexts (Theorem 5).

[3]The last assumption is necessary for the reduction to simulate a Ro$-Wrap challenge query on an already punctured tag in the find$-cpa game.

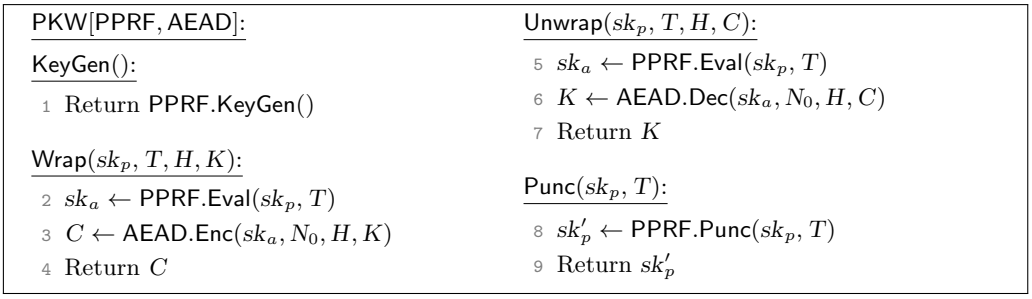

Figure 7: The PKW[PPRF, AEAD] instantiation of a puncturable key-wrapping scheme based on a puncturable pseudorandom function PPRF and a nonce-based AEAD scheme AEAD (with N_0 a fixed nonce in the nonce space of AEAD).

Lemma 1 (PKW[PPRF, AEAD] is puncture invariant and consistent)**.** *The puncturable key-wrapping scheme* PKW[PPRF, AEAD] *in Figure 7 is consistent (as per Definition 9). Additionally, if* PPRF *is puncture invariant (as per Definition 5), then* PKW[PPRF, AEAD] *is puncture invariant (Definition 8).*

Proof. The puncture invariance of PKW[PPRF, AEAD] follows immediately from the puncture invariance of PPRF. The consistency of PKW[PPRF, AEAD] follows from the correctness of the PPRF and the determinism of the encryption algorithm of the AEAD scheme. The former ensures the PPRF evaluation of a point does not change upon puncturing of other points, meaning that the AEAD key derived from a tag T is unaffected by puncturing on other tags. The latter in turn ensures that the ciphertext only depends on the inputs to AEAD encryption, none of which are affected by the puncturing of other tags.

To see this, note that by induction, point (2) of PPRF correctness (Def. 4) implies that for all subsets of elements in the domain (here the tag space of PKW) $\{T_1, T_2, \ldots, T_n\} \subset \mathcal{T}$ and all $T \in \mathcal{T} \setminus \{T_1, T_2, \ldots, T_n\}$,

$$\Pr\left[\,\mathsf{Eval}(sk_0, T) = \mathsf{Eval}(sk_n, T) \mid sk_0 \leftarrow\!\!\$\ \mathsf{PPRF.KeyGen}()\,\right] = 1,$$

where sk_n is obtained by running $sk_i \leftarrow\!\!\$\ \mathsf{PPRF.Punc}(sk_{i-1}, T_i)$ for $i \in \{1, \ldots, n\}$. Now for consistency the requirement is precisely that for all $H \in \mathcal{H}$ and all $K \in \mathcal{K}$,

$$\Pr\left[\,\mathsf{Wrap}(sk_0, T, H, K) = \mathsf{Wrap}(sk_n, T, H, K) \mid sk_0 \leftarrow\!\!\$\ \mathsf{KeyGen}()\,\right] = 1,$$

where sk_n is again obtained by recursively puncturing sk_0 on $T_1, T_2, \ldots, T_n$. This is fulfilled for PKW[PPRF, AEAD], since by definition of the construction $\mathsf{KeyGen}() := \mathsf{PPRF.KeyGen}()$, $\mathsf{Punc}(sk, T) := \mathsf{PPRF.Punc}(sk, T)$ and

$$\begin{aligned}\mathsf{Wrap}(sk_0, T, H, K) &:= \mathsf{AEAD.Enc}(\mathsf{PPRF.Eval}(sk_0, T), T, H, K)\\ &= \mathsf{AEAD.Enc}(\mathsf{PPRF.Eval}(sk_n, T), T, H, K) =: \mathsf{Wrap}(sk_n, T, H, K),\end{aligned}$$

thanks to PPRF correctness. □

Theorem 2 (PKW[PPRF, AEAD] is find$-cpa secure)**.** *Let* PKW[PPRF, AEAD] *be the puncturable key-wrapping scheme in Figure 7. For every adversary $\mathcal{A}$ against the* find$-cpa*-security of* PKW[PPRF, AEAD] *making at most q_n, $q_{ro\$}$, q_{corr} and q_p queries to oracles* New, Ro$-Wrap, Corr *and* Punc*, respectively, there exists adversaries $\mathcal{B}_{\mathsf{pprf}}$ and $\mathcal{B}_{\mathsf{aead}}$ running in approximately the same time as $\mathcal{A}$ such that*

$$\mathbf{Adv}^{\text{find\$-cpa}}_{\mathsf{PKW[PPRF,AEAD]}}(\mathcal{A}) \leq 2 \cdot \mathbf{Adv}^{\text{fpr-ro\$}}_{\mathsf{PPRF}}(\mathcal{B}_{\mathsf{pprf}}) + \mathbf{Adv}^{\text{ind\$-cpa}}_{\mathsf{AEAD}}(\mathcal{B}_{\mathsf{aead}}).$$

Adversary $\mathcal{B}_{\mathsf{pprf}}$ makes at most q_n, $q_{ro\$}$, q_{corr}, and q_p queries to oracles New, Ro$-Eval, Corr*, resp.* Punc*. Adversary $\mathcal{B}_{\mathsf{aead}}$ makes at most $q_{ro\$}$ queries to oracles* New *and* Ro$*.*

Proof. We first leverage the fpr-ro$ security of PPRF to replace the AEAD keys by random ones, then in a second step apply ind$-cpa security of AEAD to argue that wrapped PKW[PPRF, AEAD] ciphertexts are indistinguishable from random. The first step consists of a game hop from the original find$-cpa game, abbreviated $\mathbf{G}_0$, to a game $\mathbf{G}_1$ which replaces the outputs of PPRF by random AEAD keys in the implementation of oracle Ro$-Wrap. We bound the difference $|\Pr[\,\mathbf{G}_0\,] - \Pr[\,\mathbf{G}_1\,]|$ by the distinguishing advantage of an adversary $\mathcal{B}_{\mathsf{pprf}}$ against the fpr-ro$ security of PPRF (cf. Definition 6).

Adversary $\mathcal{B}_{\mathsf{pprf}}$ draws a random bit b' and acts as the challenger in game $\mathbf{G}_0$. When $b' = 1$ adversary $\mathcal{B}_{\mathsf{pprf}}$ simulates the "real world" in the PKW game, wrapping the keys output by adversary $\mathcal{A}$. When $b' = 0$, adversary $\mathcal{B}_{\mathsf{pprf}}$ simulates the "random world" and returns random strings in the ciphertext space of the AEAD scheme in response to challenge queries from $\mathcal{A}$. Finally, when adversary $\mathcal{A}$ halts and outputs bit $b^*_{\mathcal{A}}$, adversary $\mathcal{B}_{\mathsf{pprf}}$ returns 1 if $b^*_{\mathcal{A}} = b'$ and 0 otherwise.

Let b denote the random bit drawn by the challenger in the fpr-ro$ game. When $b = 1$, adversary $\mathcal{B}_{\mathsf{pprf}}$ simulates game $\mathbf{G}_0$ for $\mathcal{A}$. When $b = 0$, the simulation corresponds to game $\mathbf{G}_1$. This gives $\mathbf{Adv}^{\text{fpr-ro\$}}_{\mathsf{PPRF}}(\mathcal{B}_{\mathsf{pprf}}) = |\Pr[\mathbf{G}_0] - \Pr[\mathbf{G}_1]|$.

It remains to bound $\Pr[\mathbf{G}_1(\mathcal{A})]$. A straightforward reduction to the multi-key ind$-cpa security of AEAD gives $\Pr\left[\mathbf{G}^{\text{ind\$-cpa}}_{\mathsf{AEAD}}(\mathcal{B}_{\mathsf{aead}})\right] = \Pr[\mathbf{G}_1(\mathcal{A})]$ for an adversary $\mathcal{B}_{\mathsf{aead}}$ which simulates game $\mathbf{G}_1$ for adversary $\mathcal{A}$. Adversary $\mathcal{B}_{\mathsf{aead}}$ acts as the challenger in the game, except for when adversary $\mathcal{A}$ makes a query to oracle RO$-WRAP. To respond to such a query RO$-WRAP$(j, T, H, K)$, $\mathcal{B}_{\mathsf{aead}}$ first queries oracle NEW to initiate a new AEAD key. Additionally it increments an internal key counter i by one. It then issues a (single) query RO$$(i, N_0, H, K)$, requesting the challenge to be under the new key. The assumption that adversary $\mathcal{A}$ is tag-respecting ensures that this is a sound simulation. □

Theorem 3 ($\mathsf{PKW}[\mathsf{PPRF}, \mathsf{AEAD}]$ is find$-rcpa secure)**.** *Let* $\mathsf{PKW}[\mathsf{PPRF}, \mathsf{AEAD}]$ *be the puncturable key-wrapping scheme in Figure 7. For every adversary* $\mathcal{A}$ *against the* find$-rcpa*-security of* $\mathsf{PKW}[\mathsf{PPRF}, \mathsf{AEAD}]$ *making at most* q_n, $q_{ro\$}$, q_w q_p *and at most* q_{corr} *queries to oracles* NEW, RO$-WRAP, WRAP, PUNC *and* CORR, *respectively, there exists adversaries* $\mathcal{B}_{\mathsf{aead}}$ *and* $\mathcal{B}_{\mathsf{pprf}}$ *running in approximately the same time as* $\mathcal{A}$ *such that*

$$\mathbf{Adv}^{\text{find\$-rcpa}}_{\mathsf{PKW}[\mathsf{PPRF},\mathsf{AEAD}]}(\mathcal{A}) \leq 2 \cdot \mathbf{Adv}^{\text{fpr-rro\$}}_{\mathsf{PPRF}}(\mathcal{B}_{\mathsf{pprf}}) + \mathbf{Adv}^{\text{ind\$-cpa}}_{\mathsf{AEAD}}(\mathcal{B}_{\mathsf{aead}}).$$

Adversary $\mathcal{B}_{\mathsf{pprf}}$ *makes at most* q_n, $q_{ro\$}$, q_w, q_p, *and* q_{corr} *queries to oracles* NEW, RO$-EVAL, EVAL, PUNC, *resp.* CORR. *Adversary* $\mathcal{B}_{\mathsf{aead}}$ *makes at most* $q_{ro\$}$ *queries to oracles* NEW *and* RO$.

Proof idea. The proof follows the same strategy as the proof of Theorem 2, with the only difference that the first game hop (when the PPRF evaluations used as AEAD keys are replaced by random strings) is bounded by the advantage of an adversary against the fpr-rro$ security (cf. Definition 6) instead of the fpr-ro$ security of PPRF. Adversary $\mathcal{B}_{\mathsf{pprf}}$ uses the real evaluation oracle EVAL present in the fpr-rro$ game to simulate queries to oracle WRAP. □

Theorem 4 ($\mathsf{PKW}[\mathsf{PPRF}, \mathsf{AEAD}]$ is find$-1cpa secure)**.** *Let* $\mathsf{PKW}[\mathsf{PPRF}, \mathsf{AEAD}]$ *be the puncturable key-wrapping scheme in Figure 7. For every adversary* $\mathcal{A}$ *against the* find$-1cpa*-security of* $\mathsf{PKW}[\mathsf{PPRF}, \mathsf{AEAD}]$ *making at most* q_n *queries to oracle* NEW-RO$-WRAP, *there exists adversaries* $\mathcal{B}_{\mathsf{aead}}$ *and* $\mathcal{B}_{\mathsf{pprf}}$ *running in approximately the same time as* $\mathcal{A}$ *such that*

$$\mathbf{Adv}^{\text{find\$-1cpa}}_{\mathsf{PKW}[\mathsf{PPRF},\mathsf{AEAD}]}(\mathcal{A}) \leq 2 \cdot \mathbf{Adv}^{\text{fpr-1ro\$}}_{\mathsf{PPRF}}(\mathcal{B}_{\mathsf{pprf}}) + \mathbf{Adv}^{\text{ind\$-cpa}}_{\mathsf{AEAD}}(\mathcal{B}_{\mathsf{aead}}).$$

Adversary $\mathcal{B}_{\mathsf{pprf}}$ *makes at most* q_n *queries to oracle* NEW-RO$-EVAL. *Adversary* $\mathcal{B}_{\mathsf{aead}}$ *makes at most* q_n *queries to oracle* NEW, *and at most* q_n *queries to oracle* RO$, *of which at most one under each key.*

Proof. We first leverage the security of PPRF to replace the AEAD keys used to encrypt the session keys in the real world by random ones. We bound the difference in success probability of adversary $\mathcal{A}$ as a result of this change by the distinguishing advantage of an adversary $\mathcal{B}_{\mathsf{pprf}}$ in the fpr-1ro$ game. (Def. 6) Let $\mathbf{G}_0 := \mathbf{G}^{\text{find\$-1cpa}}_{\mathsf{PKW}[\mathsf{PPRF},\mathsf{AEAD}]}$ and let $\mathbf{G}_1$ be a game which is equivalent to $\mathbf{G}_0$, except that the AEAD keys used to encrypt the challenge session keys are drawn uniformly at random, rather than being the evaluation of PPRF on the tag chosen by $\mathcal{A}$. The games and code of adversaries for the proof are shown in Figure 8. Adversary $\mathcal{B}_{\mathsf{pprf}}$ draws a random bit b' and uses this to act as the challenger for adversary $\mathcal{A}$. To simulate the response to a query NEW-RO$-WRAP$(T, H, K)$, adversary $\mathcal{B}_{\mathsf{pprf}}$ queries oracle NEW-RO$-EVAL on input T to obtain the punctured PPRF key and a real-or-random evaluation on T. If $b' = 1$, it uses the latter as the AEAD key and encrypts K, as would the challenger in game $\mathbf{G}_0$ in the real world. If $b' = 0$, adversary $\mathcal{B}_{\mathsf{pprf}}$ simulates the ideal world and instead samples a "ciphertext" u.a.r. in $\{0,1\}^{\mathsf{cl}(|K|)}$. The PPRF key and the ciphertext are returned to adversary $\mathcal{A}$. When adversary $\mathcal{A}$ halts and outputs bit $b^*_{\mathcal{A}}$, adversary $\mathcal{B}_{\mathsf{pprf}}$ returns 1 if $b^*_{\mathcal{A}} = b'$ and 0 otherwise.

Let b denote the random bit drawn by the challenger in the fpr-1ro$ game. With the strategy described, adversary $\mathcal{B}_{\mathsf{pprf}}$ simulates game $\mathbf{G}_0$ for $\mathcal{A}$ if $b = 1$, otherwise game $\mathbf{G}_1$. Therefore

$$\mathbf{Adv}^{\text{fpr-1ro\$}}_{\mathsf{PPRF}}(\mathcal{B}_{\mathsf{pprf}}) = \left|\Pr[\mathbf{G}_0(\mathcal{A})] - \Pr[\mathbf{G}_1(\mathcal{A})]\right|.$$

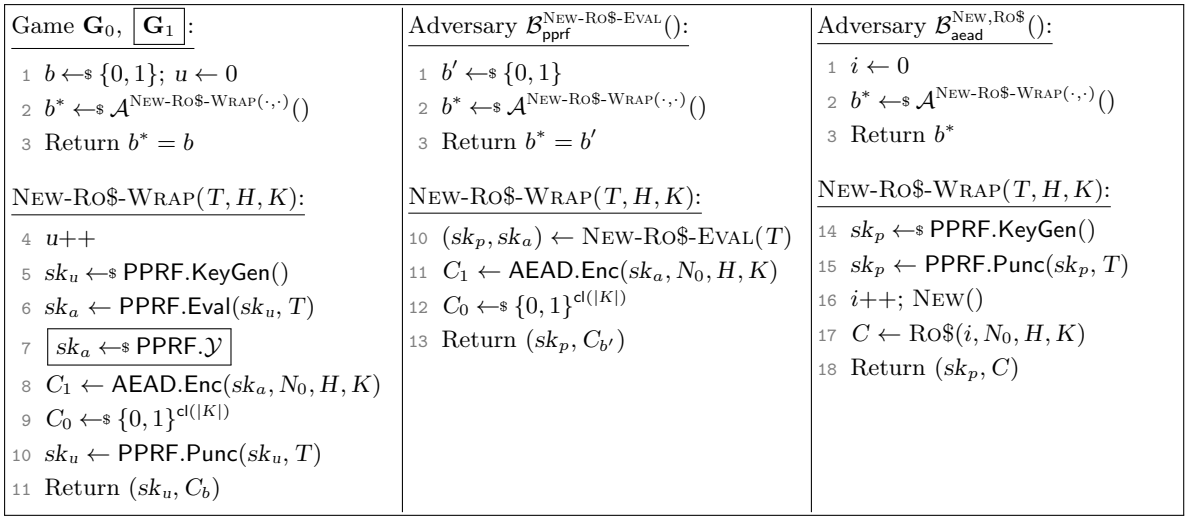

Figure 8: Games and adversaries for proof of Theorem 4.

Next, we apply confidentiality of AEAD to finish the proof. A straightforward reduction to the ind$-cpa security of AEAD gives $\Pr\left[\mathbf{G}_{\mathsf{AEAD}}^{\text{ind\$-cpa}}(\mathcal{B}_{\mathsf{aead}})\right] = \Pr\left[\mathbf{G}_1(\mathcal{A})\right]$ for an adversary $\mathcal{B}_{\mathsf{aead}}$ which simulates game $\mathbf{G}_1$ for adversary $\mathcal{A}$. To create the wrap of session key K in query New-Ro$-Wrap$(T, H, K)$, adversary $\mathcal{B}_{\mathsf{aead}}$ calls oracle New to initialize a new AEAD key. It then relays the header H and session key K from $\mathcal{A}$ to oracle Ro$ in the AEAD game under the index of the new AEAD key, letting K take the place of the message. When adversary $\mathcal{A}$ halts and returns b^*, adversary $\mathcal{B}$ also halts and returns b^*. This way, adversary $\mathcal{B}_{\mathsf{aead}}$ perfectly simulates game $\mathbf{G}_1$ for $\mathcal{A}$. Putting the two reductions together gives the theorem statement. □

Note that for all our forward indistinguishability results, *one-time* multi-user AEAD security suffices, since the uniqueness of tags means that each AEAD encryption is performed under a new key. If we wanted to allow tag-reuse to enable *batch puncturing* (cf. Footnote 2), our PKW[PPRF, AEAD] scheme would need to be instantiated with a *misuse-resistant* AEAD scheme [51] to achieve find$-cpa security. Interestingly, this straightforward modification is insufficient for find$-rcpa security: the reuse of tags across real and challenge wrap queries creates a key commitment problem which breaks the reduction. This could potentially be addressed in an idealized model, cf. [39], but we leave this to future work.

Theorem 5 (PKW[PPRF, AEAD] is int-ctxt secure)**.** *Let* PKW[PPRF, AEAD] *be the puncturable key-wrapping scheme in Figure 7. For every adversary $\mathcal{A}$ against the* int-ctxt*-security of* PKW[PPRF, AEAD] *(Def. 11) making at most q_w, q_u, q_p and q_n to oracles* Wrap, Unwrap, Punc *and* New*, respectively, there exists adversaries $\mathcal{B}_{\mathsf{aead}}$ and $\mathcal{B}_{\mathsf{pprf}}$ running in approximately the same time as $\mathcal{A}$ such that*

$$\mathbf{Adv}^{\text{int-ctxt}}_{\mathsf{PKW[PPRF,AEAD]}}(\mathcal{A}) \leq \mathbf{Adv}^{\text{fpr-ro\$}}_{\mathsf{PPRF}}(\mathcal{B}_{\mathsf{pprf}}) + \mathbf{Adv}^{\text{int-ctxt}}_{\mathsf{AEAD}}(\mathcal{B}_{\mathsf{aead}}).$$

Adversary $\mathcal{B}_{\mathsf{pprf}}$ makes at most $q_w + q_u$, q_p, and q_n queries to oracles Ro$-Eval, Punc*, resp.* New*. Adversary $\mathcal{B}_{\mathsf{aead}}$ makes at most $q_w + q_u$, q_w, and q_u queries to oracles* New, Enc*, resp.* Dec*.*

Proof. Let game $\mathbf{G}_0$ be equivalent to $\mathbf{G}^{\text{int-ctxt}}_{\mathsf{PKW[PPRF,AEAD]}}$, with the algorithms of PKW[PPRF, AEAD] implemented directly using the underlying PPRF and AEAD schemes. We begin by modifying the game to replace the AEAD keys derived by evaluating PPRF by consistent random strings. Any evaluations on punctured points are replaced with $\perp$. Call the resulting game $\mathbf{G}_1$. Any advantage change in this game hop is bounded by the advantage of an adversary $\mathcal{B}_{\mathsf{pprf}}$ against the fpr-ro$ security of PPRF. The reduction works as follows.

Adversary $\mathcal{B}_{\mathsf{pprf}}$ simulates games $\mathbf{G}_0$ and $\mathbf{G}_1$ for adversary $\mathcal{A}$, using its challenge oracle Ro$-Eval to request AEAD keys when wrapping and unwrapping session keys. It directly relays any queries to oracles New and Punc from $\mathcal{A}$ to its own corresponding oracles. When adversary $\mathcal{A}$ halts, adversary $\mathcal{B}_{\mathsf{pprf}}$ checks if $\mathcal{A}$ produced a valid forgery during the game (i.e., if the win flag has been set to true). If so, adversary $\mathcal{B}_{\mathsf{pprf}}$ returns 1 to the challenger in the fpr-ro$ game, else 0.

Let b be the hidden bit drawn by the challenger in the PPRF game. Then $\mathcal{B}_{\mathsf{pprf}}$ simulates game $\mathbf{G}_0$ for $\mathcal{A}$ when $b = 1$ and $\mathbf{G}_1$ when $b = 0$. By rewriting the advantage of adversary $\mathcal{B}_{\mathsf{pprf}}$, conditioning on

the value of b, this gives $\mathbf{Adv}^{\text{fpr-ro\$}}_{\mathsf{PPRF}}(\mathcal{B}_{\mathsf{pprf}}) = |\Pr[\,\mathbf{G}_0(\mathcal{A})\,] - \Pr[\,\mathbf{G}_1(\mathcal{A})\,]|$. Adversary $\mathcal{B}_{\mathsf{pprf}}$ makes at most $q_w + q_u$ queries to oracle RO\$-EVAL and q_p queries to oracle PUNC.

Next, we design an adversary $\mathcal{B}_{\mathsf{aead}}$ against the multi-key integrity of AEAD such that $\mathbf{Adv}^{\text{int-ctxt}}_{\mathsf{AEAD}}(\mathcal{B}_{\mathsf{aead}}) \geq \Pr[\,\mathbf{G}_1(\mathcal{A}) \Rightarrow \text{true}\,]$. Adversary $\mathcal{B}_{\mathsf{aead}}$ simulates game $\mathbf{G}_1$ for adversary $\mathcal{A}$, using oracle ENC to wrap and oracle DEC to unwrap. The key index j used by $\mathcal{B}_{\mathsf{aead}}$ in its oracle queries is determined by the PKW key index i and tag T in the query by $\mathcal{A}$. Adversary $\mathcal{B}_{\mathsf{aead}}$ keeps a table $\mathsf{T}[\cdot,\cdot]$, and each time adversary $\mathcal{A}$ makes a wrap or unwrap query under a new pair (i, T), adversary $\mathcal{B}_{\mathsf{aead}}$ calls oracle NEW to initialize a fresh AEAD key and increments j by 1. It stores the index j of the new AEAD key in $\mathsf{T}[i, T]$. In subsequent queries from $\mathcal{A}$ on (i, T), the index in $\mathsf{T}[i, T]$ is used to indicate under which AEAD key the encryption/decryption should be performed. To simulate puncturing, adversary $\mathcal{B}_{\mathsf{aead}}$ sets $\mathsf{T}[i, T]$ to $\perp$ when $\mathcal{A}$ submits query $\text{PUNC}(i, T)$. If the check passes, $\mathcal{B}_{\mathsf{aead}}$ issues query $(\mathsf{T}[i, T], N_0, H, X)$ to oracle ENC, resp. DEC.

This way, adversary $\mathcal{B}_{\mathsf{aead}}$ perfectly simulates game $\mathbf{G}_1$ for $\mathcal{A}$. Additionally $\mathcal{B}_{\mathsf{aead}}$ wins game $\mathbf{G}^{\text{int-ctxt}}_{\mathsf{AEAD}}$ precisely when adversary $\mathcal{A}$ submits a valid forgery and wins game $\mathbf{G}_1$. To see this, consider the following two possible cases for a query $\text{UNWRAP}(i, N_0, H, C)$ that makes adversary $\mathcal{A}$ win: (1) There has not been a prior query $\text{WRAP}(i, N_0, H, K)$ such that the result was C. (2) There has been a prior query $\text{WRAP}(i, N_0, H, K)$ such that the result was C, and a query $\text{PUNC}(i, T)$. In case (1), $\text{DEC}(\mathsf{T}[i, T], N_0, H, C)$ is a winning query also in game $\mathbf{G}^{\text{int-ctxt}}_{\mathsf{AEAD}}$. In case (2), the AEAD key represented by index $\mathsf{T}(i, T)$ must be $\perp$ by definition of game $\mathbf{G}_1$, meaning that the query can in fact not be a successful forgery. Hence this case is void.

This shows that $\Pr\left[\,\mathbf{G}^{\text{int-ctxt}}_{\mathsf{AEAD}}(\mathcal{B}_{\mathsf{aead}})\,\right] \geq \Pr[\,\mathbf{G}_1(\mathcal{A})\,]$. Together with the bound on the first game hop, this gives the theorem statement. □

5 TLS Ticketing

Equipped with a secure instantiation of a puncturable key-wrapping scheme, we now turn our attention to applications and begin with the Transport Layer Security (TLS) protocol. We show how the ticketing approach taken in its resumption handshake protocol can be instantiated with a PKW scheme, increasing forward security of resumed sessions.

A TLS connection between clients and servers begins with the establishment of a shared symmetric key through a so called *handshake*. For repeated connections, TLS offers a *resumption* handshake mode with better performance, which bootstraps security from a *pre-shared key* (PSK) established in a prior full handshake. In TLS 1.3 [49], this is referred to as the *PSK mode*.

In order to enable a resumption handshake, the so-called "resumption master secret" RMS is derived in a TLS 1.3 handshake and then used to derive (usually multiple) pre-shared keys for later resumptions. For each such pre-shared key, the TLS 1.3 server sends the client a unique nonce N_T, and both derive the pre-shared key as $\text{PSK} \leftarrow \mathsf{HKDF.Expand}(\text{RMS}, \texttt{"tls13 resumption"} \| N_T)$ using the HKDF key derivation function [42]. The client will store all PSKs established, but the server may outsource this storage to the client, e.g., by encrypting PSK under a long-term symmetric key, the so-called Session Ticket Encryption Key (STEK), and sending the resulting ciphertext (as the PSK identifier) to the client. This process of outsourcing the server-side resumption state to the client is commonly referred to as *ticketing* [53], and the identifier hence called a *ticket*.

One issue with TLS ticketing is that the tickets are generally not forward secret: if an attacker compromises the STEK, it will be able to recover the PSKs encrypted in prior resumption handshakes, thereby compromising the security of the concerned sessions. While TLS 1.3 allows for ephemeral Diffie–Hellman secrets to be mixed into the key derivation, the so-called "early" or "zero round-trip time" (0-RTT) data that a client can send immediately does not enjoy this protection, and hence would be exposed if the PSK were to be compromised.

Aviram, Gellert, and Jager (AGJ) [2] recently proposed an approach to achieve forward-secure session ticketing, giving forward security even for 0-RTT data, through what they call "session resumption protocols." In this section we revisit their approach and show how their session resumption mechanism can be viewed more simply through the lens of puncturable key wrapping: First of all, their construction is mimicked by our instantiation $\mathsf{PKW}[\mathsf{PPRF}, \mathsf{AEAD}]$ of a PKW from a puncturable PRF and an AEAD scheme, when tags are chosen (and sent as part of the TLS ticket) as *counters*. More importantly, capturing TLS ticketing through the PKW scheme $\mathsf{PKW}[\mathsf{PPRF}, \mathsf{AEAD}]$ allows us to seamlessly switch to a more *privacy-friendly* variant: by choosing the tags as random values, we make the entire TLS ticket random-looking. This avoids the potentially traceable counter element in the AGJ [2] ticketing

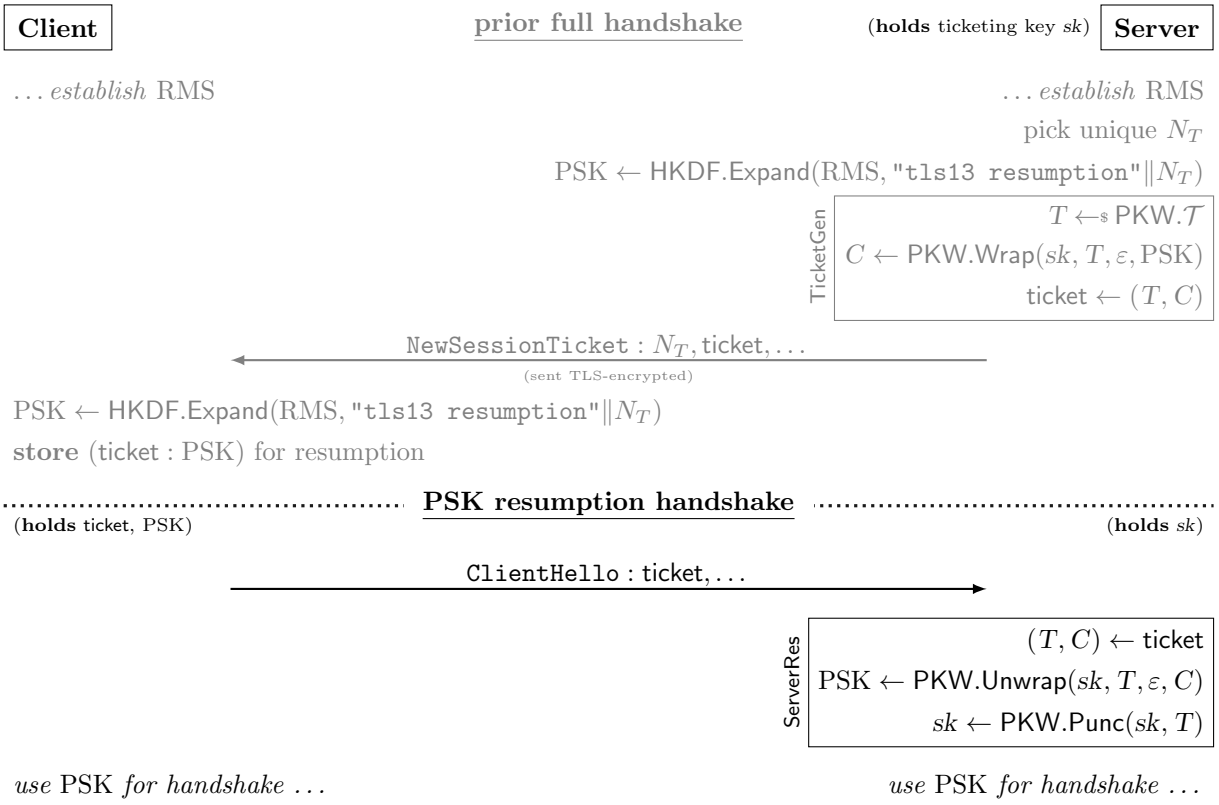

Figure 9: Forward-secure TLS 1.3 0-RTT pre-shared key (PSK) resumption handshake using a puncturable key-wrapping scheme PKW (bottom part), based on a session ticket generated by the server and stored by the client in a prior full handshake (upper part, in gray). The boxed sections can be read as the PKW-based instantiation of a session resumption protocol [2], with tag sampling and wrapping corresponding to ticket generation (TicketGen) and unwrapping and puncturing corresponding to session resumption (ServerRes); the PKW key *sk* plays the role of the STEK.

proposal, thereby addressing privacy concerns for TLS ticketing, e.g., regarding tracking users on the web by passive network observers (see [57] for a broader discussion).

When rephrasing the AGJ integration of a session resumption protocol into the TLS 1.3 resumption handshake [2, Section 4.2, 4.3] as puncturable key wrapping, we found conceptual and technical issues in their proposed protocol, the security model, and the proof. These prevent their proposal from being (forward-)secure as-is. We rectify this situation through the following corrections:

1. Ticketing the right key. In AGJ, the TLS 1.3 resumption master secret RMS is encrypted in the session ticket(s). However, RMS is used to derive *multiple* pre-shared keys PSK for resumption. Ticketing RMS thus violates the goal of forward security: an adversary learning RMS from one ticket can use that value to decrypt prior sessions using a PSK derived from the same RMS.

   In our protocol integration (cf. Figure 9), we instead ticket PSK, not RMS, following the TLS 1.3 RFC [49, Section 4.6.1].

2. Accurately modeling tickets and corruption. The security model in AGJ does not reflect the ticketing mechanism of a key exchange protocol in how pre-shared secrets are sampled, registered with parties, and possibly corrupted. This leads to their model, strictly speaking, being unable to capture the ticketing mechanism of TLS resumption.[4] Only allowing server-side corruptions, their model also fails to capture that an adversary might compromise pre-shared secrets stored by clients.

   In our security model, we integrate the protocol's ticketing mechanism and allow the adversary to corrupt both the ticketing mechanism keys of servers, as well as stored secrets of clients.

[4] E.g., when setting up new pre-shared keys, their model takes the identifier *psid* of the key as an *adversary-provided* input, while *psid* in fact corresponds to the ticket *(honestly) output* by the protocol's ticketing mechanism. This means that their model is actually unable to capture how tickets are generated by (honest) servers.

3. Rectifying proof steps. The security proof for the protocol integration of AGJ [2, Theorem 4] only uses part of the power of their session resumption primitive (i.e., a single challenge where their primitive provides many), and also misses some preliminary steps (esp. the necessity of puncture invariance and consistency, which our PKW formalism brings to light).

   In our proof, we add these missing steps and show that reducing to the weaker one-time PKW security suffices for our integration.

4. Making underlying assumptions precise. The AGJ proof makes two undefined assumptions on the underlying session resumption resp. PPRF scheme. Formally, this leads to an issue with the security proof of their construction, which in turn enables a theoretical violation of the formal integrity claims on their protocol. We give the details in Appendix D.

   Through our formalism for puncturable key wrapping and PPRFs, we make the necessary assumptions (puncture invariance for PKW, resp. demanding $\perp$ output after puncturing for PPRFs) visible and explicit.

Overall, our exposition stays close to the approach by AGJ, focusing on the necessary corrections. We see this not only as an illustration that puncturable key wrapping is readily applicable to achieve forward-secure 0-RTT session resumption, but also that this conceptual framework helps to avoid errors when integrating puncturing techniques into more complex applications.

5.1 Integration into the TLS 1.3 Handshake

The integration of puncturable key wrapping into the TLS 1.3 resumption handshake for achieving forward-secure 0-RTT resumption is illustrated in Figure 9. When issuing a ticket in a full handshake (upper part of Figure 9), the server picks a tag T, wraps the to-be-used pre-shared key PSK using the PKW scheme under that tag, and sends the tag and resulting ciphertext as the ticket. (This corresponds to the ticket issuing algorithm TicketGen of AGJ in [2, Fig. 4].) Upon resumption (Figure 9, bottom part), the client will send this ticket to the server, which the server can use to first unwrap PSK and then puncture on the tag to achieve forward security. (This corresponds to the session resuming algorithm ServerRes in [2, Fig. 4].) We remark that while in our abstraction, we set the PKW header to be empty (ε), this field may in practice be used to authenticate further context of the TLS ticket, like its belonging to a TLS 1.3 handshake or the name of the server.

Enhancing privacy. We note that for fine-grained security when puncturing on a per-ticket level, the tag T must not repeat. In the generic construction of a session resumption protocol from a PPRF and AEAD scheme by AGJ [2, Construction 1], a counter takes the place of the tag in our representation, incremented with every issued ticket. While this ensures uniqueness, it makes subsequently issued tickets to the same user linkable and leaks the order in which users return to a server, which may lead to privacy concerns.[5]

We therefore suggest employing a *random* tag T, as this makes the overall ticket a random-looking, single-use string. In case availability of (true) randomness is of concern, this tag may be sampled through a randomly-seeded PRG chain. Through our unified, tag-based interface for PKW schemes, this switch to a more privacy-friendly ticket generation is seamless. The only disadvantage is the possibility of collisions amongst the randomly chosen tags; this requires us to work with a larger tag space, but does not overly affect efficiency. For example, in the concrete GGM-based construction for PPRFs (and hence PKWs), the tree-depth grows linearly with the bit-length of the tags and the tag length needs to be roughly doubled (compared to the counter-based approach of AGJ) to keep the collision probability small. Moreover, if key wrapping fails during ticket generation (because the secret key has already been punctured at the chosen tag), the server can simply re-sample the tag.

Ensuring forward security. The careful reader might have observed that in the protocol TLS13wRES of AGJ [2, Fig. 4], the integration of session resumption into the TLS 1.3 handshake involves generation of a ticket on the resumption master secret (RMS) RMS (and N_T), while in our integration in Figure 9, the pre-shared key PSK derived from RMS is wrapped, rather than RMS itself. Our choice of ticketing (or: wrapping) PSK instead of RMS is deliberate: it corrects a weakness in the AGJ TLS13wRES protocol which results in the loss of forward security and which actually invalidates their proof.

[5] Sy et al. [57] discuss how tickets, sent in the clear upon resumption, can be exploited for tracking TLS 1.3 users on the web.

To see this, recall that in TLS 1.3, RMS is used in a full handshake to regularly derive *multiple* pre-shared keys PSK (using different nonces N_T, which RFC 8446 [49] only requires to be unique, not secret), allowing the client many resumption connections. An attacker that gets hold of one of these tickets and compromises the server's ticketing secret key sk (towards breaching forward security), can then recover RMS. From there it is possible to derive other PSK values ticketed from RMS in the same original handshake, *even if* those other PSK tickets have been processed and punctured on—violating their forward security.

We address this issue by ticketing PSK and not RMS. This actually follows the TLS 1.3 RFC [49, Section 4.6.1] which notes that the association ought to be "between the ticket value and a secret PSK derived from the resumption master secret."

5.2 Security Model

AGJ [2] study the composition of their session resumption protocol with the TLS 1.3 resumption handshake through a multi-stage key exchange (MSKE) model, slightly adapting the model introduced by Fischlin and Günther [30, 31]. This modeling reflects that the TLS 1.3 resumption handshake establishes multiple keys with varying security properties. In particular, it allows to capture the intended forward security of 0-RTT keys when deploying session tickets based on the session resumption protocol by AGJ.

In a multi-stage key exchange model, an attacker can set up new pre-shared secrets (randomly sampled by a challenger through a NewSecret oracle) and instruct parties to initiate new sessions using those secrets (via a NewSession oracle). The adversary is given the ability to eavesdrop on and actively manipulate (via a Send oracle) the exchanged messages between many protocol participants running, concurrently, multiple executions of the protocol. The attacker is further allowed to compromise the long-term secrets of participants (via oracle Corrupt) and to reveal the established session key in stages of sessions of its choice (via Reveal). Security is then defined in the sense of session key indistinguishability: the attacker should be unable to tell apart the real session key in a not trivially compromised ("fresh") session from a random key, obtained through a Test oracle.

We concur with the approach of AGJ [2] to capture the security of forward-secure TLS resumption as a multi-stage key exchange protocol. However, we observe that their model requires changes to be able to indeed capture their proposed protocol, and its envisioned forward security. In the following, we discuss the core changes needed; as the main parts of the model remain unchanged, we only summarize it on a high level and refer to AGJ [2] for details.

Associating users and long-term keys. Servers issuing tickets require their long-term keys to be *updated* during protocol execution and *new PSK registrations*. The latter aspect is missing in the AGJ model: Technically, the adversary is not given access to the actual ticket generation mechanism of the protocol, a feature clearly required to accurately model the protocol.

We accordingly revise the NewSecret oracle (cf. Figure 10) in AGJ to, beyond sampling a new random pre-shared secret pss, register this key with both involved parties. The latter is done via an auxiliary algorithm RegisterSecret defined by the protocol which captures how pre-shared secrets are stored by both participants. Without ticketing, RegisterSecret would plainly store an association between $psid$ and pss; this is what prior MSKE models on TLS 1.3 captured [30, 25, 31, 26]. For ticketing, RegisterSecret encodes the ticket generation procedure missing in the AGJ model; Figure 10 defines it for our instantiation via a puncturable key-wrapping scheme PKW.

Corruptions can also compromise clients. Furthermore, the AGJ model did not allow (client) secret keys to be corrupted (through oracle Corrupt), unnecessarily resulting in a weaker security model which does not capture the effects of such compromise. To remedy this, we modify the oracle Corrupt(U) from AGJ to return pss_U, allowing the adversary to also comprise client-side stored secrets. In our instantiation, this corresponds to leaking the PKW secret key srk_U of a server U (as before in AGJ) and leaking the stored tickets and pre-shared secrets $pss_U[V, psid] = pss$ of a client U (new). (Non–forward-secure stages of sessions that used the leaked keys are set to be revealed.)

5.3 Security Proof

With the revised model in place, we now revisit the security proof of AGJ for integration of session ticketing into TLS 1.3. We will do so for our adapted instantiation based on a PKW scheme PKW, given

NewSecret(U, V): // Set up secret between initiator U and responder V

1 $pss \leftarrow\!\!\$$ PSSSPACE // sample at random from pre-shared secret space

2 $(psid, sk_U, sk_V) \leftarrow$ RegisterSecret(U, V, sk_U, sk_V, pss)

3 return $psid$

RegisterSecret$(U, V, sk_U = (pss_U, srk_U), sk_V = (pss_V, srk_V), pss)$:

1 $T \leftarrow\!\!\$$ PKW.$\mathcal{T}$ // sample random puncturing tag for ticket

2 $C \leftarrow$ PKW.Wrap$(srk_V, T, \varepsilon, pss)$ // wrap pss under V's PKW key srk_V

3 $psid \leftarrow (T, C)$ // $psid$ is the ticket; it consists of tag and wrapping ciphertext

4 $pss_U[V, psid] \leftarrow pss$ // add pss to U's list of keys shared with V

5 return $(psid, (pss_U, srk_U), (pss_V, srk_V))$

Figure 10: Revised NewSecret oracle and auxiliary algorithm RegisterSecret capturing our TLS ticketing instantiation via a puncturable key-wrapping scheme PKW.

in Figure 9. Our exposition follows the proof structure of AGJ (cf. [2, Section 4.3] for its details), but focuses on highlighting the necessary changes and corrections.

Theorem 6 (TLS 1.3 with PKW ticketing security (informal))**.** *The TLS 1.3 resumption protocol with session ticketing based on a PKW scheme* PKW *as depicted in Figure 9 is a secure multi-stage key exchange (MSKE) protocol (with forward security from the first stage on), if* PKW *is* puncture invariant*,* consistent*, and* forward indistinguishable under a one-time challenge (find$-1cpa)*, the further involved hash function* H *is collision resistant, and the HKDF extraction and expansion steps satisfy PRF security.*

Formally, the advantage of a multi-stage adversary $\mathcal{A}$ against the protocol in Figure 9 is bounded as:

$$\mathbf{Adv}^{\text{MSKE}}_{\text{TLS[PKW]}}(\mathcal{A}) \leq 5n_s \cdot \Big(\mathbf{Adv}^{\mathsf{coll}}_{\mathsf{H}}(\mathcal{B}_1) + n_u \cdot n_{t/u} \cdot \Big(\mathbf{Adv}^{\text{find\$-1cpa}}_{\mathsf{PKW}}(\mathcal{B}_2) + \mathbf{Adv}^{\mathsf{dual\text{-}PRF}}_{\mathsf{HKDF.Extract}}(\mathcal{B}_3) + 2 \cdot \mathbf{Adv}^{\mathsf{PRF}}_{\mathsf{HKDF.Extract}}(\mathcal{B}_4) + 9 \cdot \mathbf{Adv}^{\mathsf{PRF}}_{\mathsf{HKDF.Expand}}(\mathcal{B}_5) \Big) \Big),$$

where n_s is the maximum number of sessions, n_u the maximum number of users, and $n_{t/u}$ the maximum number of tickets issued by any user in the key exchange game.

Proof summary. The proof structure follows that of AGJ [2, Theorem 5], adapted to our PKW instantiation and revising some steps based on the TLS 1.3 analysis by Dowling et al. [26]. It proceeds via a series of game hops, starting with the original multi-stage key exchange game for the protocol in Figure 9, $\mathbf{G}_0 = \mathbf{G}^{\text{MSKE}}_{\text{TLS[PKW]}}$.

1. $\mathbf{G}_0 \to \mathbf{G}_1$: Hybrid argument.
   The first hop, as in AGJ [2], restricts the adversary $\mathcal{A}$ to a single Test query, fixing the test session in advance at a guessing loss of $5 \cdot n_s$, accounting for the 5 stages and up to n_s sessions. The hybrid is detailed in [26, Appendix A].

2. $\mathbf{G}_1 \to \mathbf{G}_2$: Hash collisions.
   The next hop rules out hash collisions, reducing to the hash function H's collision resistance and inducing the additive term $\mathbf{Adv}^{\mathsf{coll}}_{\mathsf{H}}(\mathcal{B}_1)$, as in AGJ.

3. $\mathbf{G}_2 \to \mathbf{G}_3$: Guessing the involved server identity V and ticket index $psid$.
   Beyond guessing the ticket (index) used in the (single) test session, as done in AGJ, in this game hop we *also* need to guess the server V involved in the test session. The latter is missing in AGJ, but crucially needed; in particular to be able to simulate ticket issuing by that server V *before* the test session's participants are known. This guessing results in a combined loss of $n_u \cdot n_{t/u}$.

4. $\mathbf{G}_3 \to \mathbf{G}_4$: Replacing the test session ticket.
   This is the core game hop involving the ticketing mechanism's security (i.e., session resumption protocol security in AGJ, and the PKW scheme's forward indistinguishability in our instantiation), and also where our corrections to the security model become most visible.

   In the AGJ proof (Game 4 and reduction $\mathcal{B}_2$ in [2, Proof of Theorem 4]), this game is defined to replace the server's ticket resumption (ServerRes) in both the tested session and its intended partner session. However, the latter is not known and—more importantly—only server sessions execute

ServerRes while clients use the stored pre-shared secret directly. The reduction $\mathcal{B}_2$ then obtains several (μ) resumption tickets through its game, but only uses the first ticket as the challenge for the tested session; the handling of further ticket issuing by the involved server is missing.

We rectify this proof step, applying the PKW scheme's forward indistinguishability find$-1cpa (one-time instead of many-challenge), puncture invariance (invoked but not defined for session resumption protocols in AGJ), as well as consistency (a missing aspect in AGJ). Our Game $\mathbf{G}_4$ consists of replacing wrapped key C inside the ticket ticket used in the test session by a uniformly random string. We reduce the advantage difference of $\mathcal{A}$ induced by this change to the PKW scheme's find$-1cpa security via the following reduction $\mathcal{B}_2$:

(a) Our reduction $\mathcal{B}_2$ embeds the find$-1cpa PKW instance at the guessed server V, not sampling that server's PKW secret key in the reduction itself anymore.

(b) At the outset of the game, $\mathcal{B}_2$ issues a single challenge query NEW-RO$-WRAP$(T, \varepsilon, K)$ (for a single user) for uniformly random tag T and key K, with empty header $H = \varepsilon$. Key K will later be embedded as the pre-shared secret into the test session. In return, the reduction $\mathcal{B}_2$ obtains a real-or-random challenge wrapping C_b which will serve as part of the ticket ticket (resp., the ticket identifier, *psid*) in the test session, as well as the corrupted secret key *sk*, punctured on T.

(c) The reduction now uses *sk* in place of the guessed server V's PKW key, in particular to issue further tickets itself. *Consistency* and *puncture invariance* ensure that the resulting tickets and puncturing (revealed via a potential later compromise of V's key) are consistent despite the challenge ticket being computed ahead of time in the prior step.

(d) When the test session's ticket is issued via NEWSESSION (guessed in the prior game hop), the reduction $\mathcal{B}_2$ embeds K as the PSK and (T, C_b) as the ticket ticket (and ticket identifier *psid*) in the test session (as well as its partner, when that becomes known).

(e) The reduction $\mathcal{B}_2$ outputs whether $\mathcal{A}$ wins in the key exchange game as its own guess for the challenge bit b. Depending on b, it simulates either $\mathbf{G}_3$ or $\mathbf{G}_4$, meaning that $\mathcal{A}$'s advantage difference between the two is bounded by $\mathbf{Adv}^{\text{find\$-1cpa}}_{\mathsf{PKW}}(\mathcal{B}_2)$.

Observe that as a result of Game $\mathbf{G}_4$, the ticket ticket sent in the tested session is now *decoupled* from the used pre-shared secret PSK in that session: both are independently drawn, uniformly random values. This will allow us to, from here, proceed via a sequence of PRF security game hops (as in the AGJ proof) to establish key indistinguishability of the derived session keys. We follow the notation of AGJ and Dowling et al. [26] for these final steps and highlight only the relevant changes compared to AGJ.

5. $\mathbf{G}_4 \to \mathbf{G}_5$: Omitted: In our instantiation, we wrap the PSK directly, so the game hop in which AGJ move from PSK being derived from RMS to PSK being uniformly random becomes unnecessary. Recall that the AGJ protocol integration issued tickets on the TLS resumption master secret RMS instead of the pre-shared key PSK, and that we correct this to ensure that forward security is actually achieved. Formally, the previous game hop makes the test session's ticket and PSK independent, which would not be the case if RMS was wrapped in the ticket: an adversary could have server V issue two tickets on RMS, one used for the test session, then compromise V's secret key and decrypt the other ticket to reveal RMS and from that, distinguish the tested key from random. This attack violates the claimed forward security of session keys in AGJ.

6. $\mathbf{G}_5 \to \mathbf{G}_{17}$: Replacing derived keys with random keys, one at a time.
From the independent and random pre-shared key PSK, we can now derive the session keys in the TLS resumption protocol through a sequence of twelve PRF (or dual-PRF[6]) game hops. For readability, we summarize the 2, resp. 9, advantage terms for PRF security of HKDF.Extract, resp. HKDF.Expand, under combined reductions $\mathcal{B}_4$, resp. $\mathcal{B}_5$. □

[6] In place of the rather ad-hoc "HMAC$(0, \$)$-$\$$" assumption on HKDF.Extract deployed in AGJ [2], originating from [31], we use the dual-PRF assumption also used in [26] to indicate PRF security of HKDF.Extract when keyed through the second input.

6 Protected File Storage

Motivated by the ubiquitous outsourcing of data storage by private individuals and companies alike, we now turn our focus to our second application, file storage, and show how a PKW scheme can be used to provide (forward) security for remotely stored sensitive data. To this end, we design a *protected file storage* (PFS) system, which provides an interface for local encryption, decryption, and secure file shredding to a privacy-concerned user. The system is inspired by the internals of cloud storage services like AWS [4], Google Cloud [34], IBM Cloud [38] and Microsoft Azure [46], but the final primitive is oblivious to the actual relationship between data owner and storage provider: in a PFS system, all trust lies with the holder of the secret key. This means that our system can cater both to users who wish to maintain control over the security of their data (and therefore retain the hold of the secret key) while offloading storage, *and* to storage providers who perform data encryption as a service.

The PFS interface is aimed at the former case, and hence hides internals of the system such as the key hierarchy to minimize the risk of involuntary misuse by an end user. However, it is still designed to support commonplace attributes of cloud storage systems, such as functionality for key rotation, as well as additionally providing fine-grained forward security for deleted files. This makes our approach conformable for use also by cloud service providers who wish to enhance the security guarantees in their existing systems.

6.1 PFS Syntax

We envision a PFS system to be utilized by a user who holds a set of (plaintext) files that they wish to protect and outsource the storage of to some storage service (e.g. a cloud). The user generates a local secret key sk via the setup algorithm $\mathsf{Setup}()$. They can then encrypt and decrypt files via algorithms $\mathsf{EncFile}$ and $\mathsf{DecFile}$, where encrypted files are associated with an identifier id, a header h, and a ciphertext C, of which the user stores h and C under the "filename" id at the storage service. (The user may keep a local look-up table mapping human-readable filenames to identifiers id, or decide to offload this table as yet another protected file to the storage service, too. In the latter case, the user only needs to store the identifier of the mapping file.) To shred a file, it suffices to locally run the algorithm $\mathsf{ShredFile}(sk, id)$ on the file identifier to be shredded. This will ensure that the corresponding file is irrecoverable (*forward secure*) from this point on; remote deletion at the service provider is not required to ensure its forward security. Finally, a user may rotate its secret key (e.g., for regulatory purposes or to refresh the key once its usage limit has been reached), which is done through calling a RotKey algorithm, taking the current list of file identifiers and headers as input and updating them with new headers to be replaced at the storage provider. Formally, a PFS scheme has the following syntax.

Definition 12 (PFS scheme)**.** A *protected file storage* scheme $\mathsf{PFS} = (\mathsf{Setup}, \mathsf{EncFile}, \mathsf{DecFile}, \mathsf{ShredFile}, \mathsf{RotKey})$ is a 5-tuple of algorithms with four associated sets; the secret key space $\mathcal{SK}$, the file space $\mathcal{F}$, the file identifier space $\mathcal{I}$, and the header space $\mathcal{H}$. Associated to the PFS is a ciphertext-length function $\mathsf{cl}\colon \mathbb{N} \to \mathbb{N}$.

- Via $sk \leftarrow\!\!\$\ \mathsf{Setup}()$, the probabilistic setup algorithm Setup, taking no input, produces a secret key $sk \in \mathcal{SK}$.
- Via $(id, h, C)/\bot \leftarrow\!\!\$\ \mathsf{EncFile}(sk, F)$, the randomized file encryption algorithm $\mathsf{EncFile}$ on input the secret key $sk \in \mathcal{SK}$ and a plaintext file $F \in \mathcal{F}$ produces a file identifier $id \in \mathcal{I}$, a header $h \in \mathcal{H}$ and a ciphertext $C \in \{0,1\}^{\mathsf{cl}(|F|)}$ or, to indicate failure, $\bot$.
- Via $F/\bot \leftarrow \mathsf{DecFile}(sk, id, h, C)$, the deterministic file decryption algorithm $\mathsf{DecFile}$ on input the key $sk \in \mathcal{SK}$, a file header $h \in \mathcal{H}$, and a ciphertext $C \in \{0,1\}^*$ returns a file plaintext $F \in \mathcal{F}$ or, to indicate failure, $\bot$.
- Via $sk' \leftarrow \mathsf{ShredFile}(sk, id)$, the deterministic file shredding algorithm $\mathsf{ShredFile}$ on input the secret key $sk \in \mathcal{SK}$ and a file identifier $id \in \mathcal{I}$ returns the updated secret key $sk' \in \mathcal{SK}$.
- Via $(sk', (h'_1, \ldots, h'_\ell))/(sk', \bot) \leftarrow\!\!\$\ \mathsf{RotKey}(sk, ((id_1, h_1), \ldots, (id_\ell, h_\ell)))$, the randomized key-rotation algorithm RotKey on input the secret key $sk \in \mathcal{SK}$ and a list of file identifier-header pairs $(id_1, h_1), \ldots, (id_\ell, h_\ell) \in (\mathcal{I} \times \mathcal{H})^*$ returns the potentially updated secret key $sk' \in \mathcal{SK}$ and a sequence of updated headers $(h'_1, \ldots, h'_\ell) \in \mathcal{H}^*$ or, to indicate failure, $\bot$.

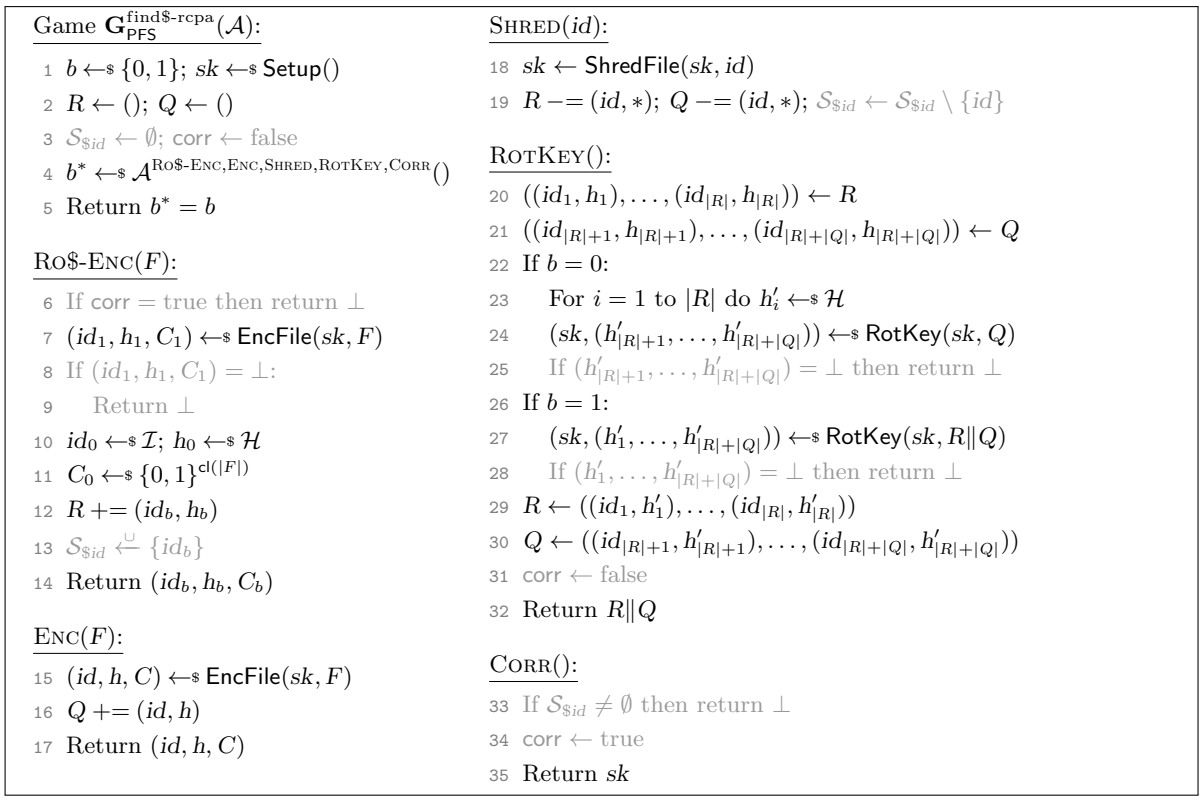

Figure 11: Confidentiality and forward security (find$-rcpa) game for a protected file storage scheme PFS. Grey code prevents trivial attacks. Lists R and Q keep track of file identifiers and headers currently in the system for the sake of key rotation. We write $L \mathrel{-}= (id, *)$ to denote removing all tuples of which the first element equals id from a list L.

Correctness. The functionality that we require from a protected file storage system is that non-shredded files can be correctly decrypted, even after key rotations. Somewhat more formally, we call a PFS scheme *correct* if decryption of a file ciphertext works with overwhelming probability, even after multiple key rotations, as long as the header has been updated in each key rotation and the file has not been shredded at any point.

We note that although one could in theory "delete" files from the system by omitting them as input when rotating the key, the intention is for the entire list of current file identifiers and headers to be used as input to RotKey. For forward security, the shredding algorithm should be used.

6.2 Confidentiality and Integrity of PFS

Confidentiality and forward security. A protected file storage scheme should provide confidentiality of the stored files, including their metadata (file identifiers and headers), as well as forward security when files have been shredded. Additionally, key rotation should allow the scheme to recover from corruption, ensuring security of newly encrypted files.

We capture this form of confidentiality through the notion of *forward indistinguishability from random bits* under *real and chosen-plaintext attack* (find$-rcpa). In the find$-rcpa security game, given in Figure 11, the adversary is asked to distinguish real from random outputs of a challenge real or $ encryption oracle Ro$-Enc. We emphasize that indistinguishability here encompasses *both* the file ciphertext *and* metadata (i.e., identifier and header), encoding a strong form of *privacy*. The game further allows the adversary to shred files (via the oracle Shred) and to rotate keys (via RotKey), leading to an update of the headers of all non-shredded files. We encode forward security via a Corr oracle, through which the adversary may ultimately learn the user's current secret key, provided that it shredded all challenge files (to prevent trivial distinguishing attacks) and does not make further challenge queries on that key. Furthermore, we allow new challenge queries *after* a successful key rotation, which captures security being regained after key rotation in which the adversary remained passive, a form of *post-compromise security* [22]. In order to capture potential leakage from unshredded files in the system which a real-world adversary would gain access to when corrupting a user's secret key, the game

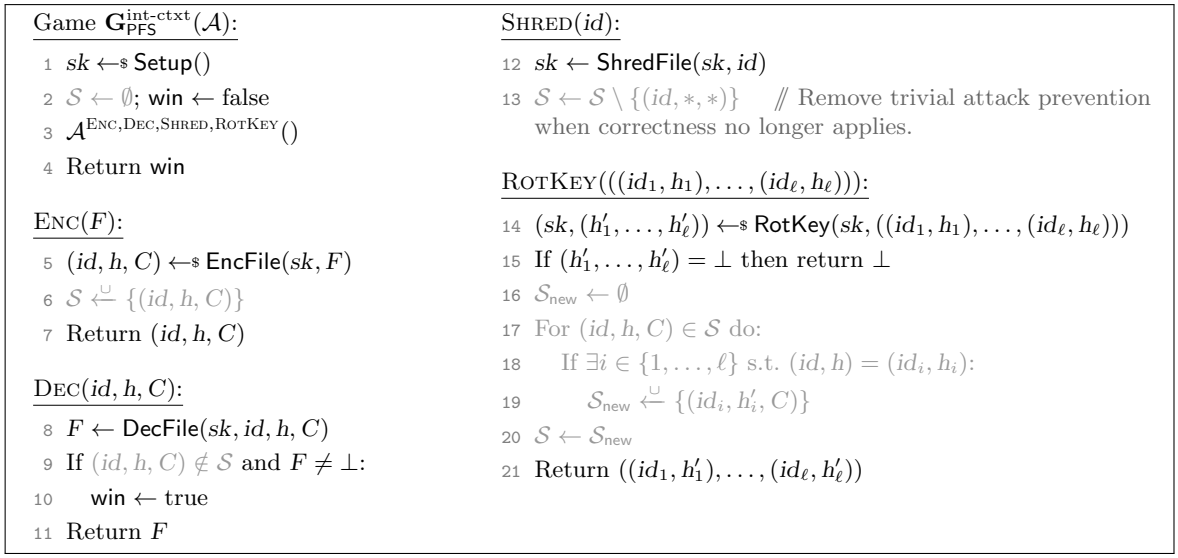

Figure 12: Integrity of ciphertexts game for a protected file storage scheme PFS. Grey code prevents trivial attacks.

additionally includes a real encryption oracle Enc, which provides the adversary with honest encryptions of plaintexts of its choice that do not need to be shredded prior to corruption.

Definition 13 (PFS confidentiality (find$-rcpa))**.** Let PFS be a protected file storage scheme and $\mathbf{G}^{\text{find\$-rcpa}}_{\mathsf{PFS}}$ be the game defined in Figure 11. We define the advantage of an adversary $\mathcal{A}$ against the find$-rcpa security of PFS as

$$\mathbf{Adv}^{\text{find\$-rcpa}}_{\mathsf{PFS}}(\mathcal{A}) = 2\left|\Pr\left[\,\mathbf{G}^{\text{find\$-rcpa}}_{\mathsf{PFS}}(\mathcal{A}) \Rightarrow \text{true}\,\right] - \frac{1}{2}\right|.$$

Integrity. In addition to confidentiality and privacy, a PFS scheme should also provide integrity of ciphertexts for the files in the system. We define this notion via the game in Figure 12. The adversary's goal here is to come up with a file tuple (id, h, C) that was not output by the encryption oracle Enc, or has been shredded (using oracle Shred), yet successfully decrypts (in the decryption oracle Dec). The game further provides access to a key rotation oracle RotKey; in contrast to the find$-rcpa game, this is strengthened to take adversarially-chosen file identifiers and headers as input. This captures that a malicious storage service might inject forged identifiers and headers into a user's storage or omit files from key rotation.

Definition 14 (PFS integrity (int-ctxt))**.** Let PFS be a protected file storage scheme and $\mathbf{G}^{\text{int-ctxt}}_{\mathsf{PFS}}$ be the game defined in Figure 12. We define the advantage of an adversary $\mathcal{A}$ against the int-ctxt security of PFS as

$$\mathbf{Adv}^{\text{int-ctxt}}_{\mathsf{PFS}}(\mathcal{A}) = \Pr\left[\,\mathbf{G}^{\text{int-ctxt}}_{\mathsf{PFS}}(\mathcal{A}) \Rightarrow \text{true}\,\right].$$

On human-friendly filenames. The file identifiers generated by a PFS system are required to be indistinguishable from random bits in order to give maximum metadata privacy against a curious cloud service provider. This security guarantee comes with the drawback that filenames are not freely choosable by the user, which can make file management less intuitive. As mentioned above, a potential remedy to this is to let the application layer running on top of the PFS system maintain an internal mapping between user-specified filenames and the file identifiers generated by the system. The mapping can be made forward-secure if it is placed in a special file and encrypted by the PFS system. Whenever a file in the PFS is shredded, the file id of the shredded file and the corresponding filename in the mapping file is deleted and the mapping encrypted anew. The old version of the mapping file can then be shredded, so that the plaintext filename of the shredded file is hidden even in the event of a future key compromise.

We remark that this kind of naive solution is directly supported by the PFS system as presented. If one would wish to achieve the same functionality in a more efficient way, the system could be tailored to treat the mapping file differently from other files in the system. For example, by using a special separate key to encrypt the mapping file, forward security could be achieved for filenames of shredded

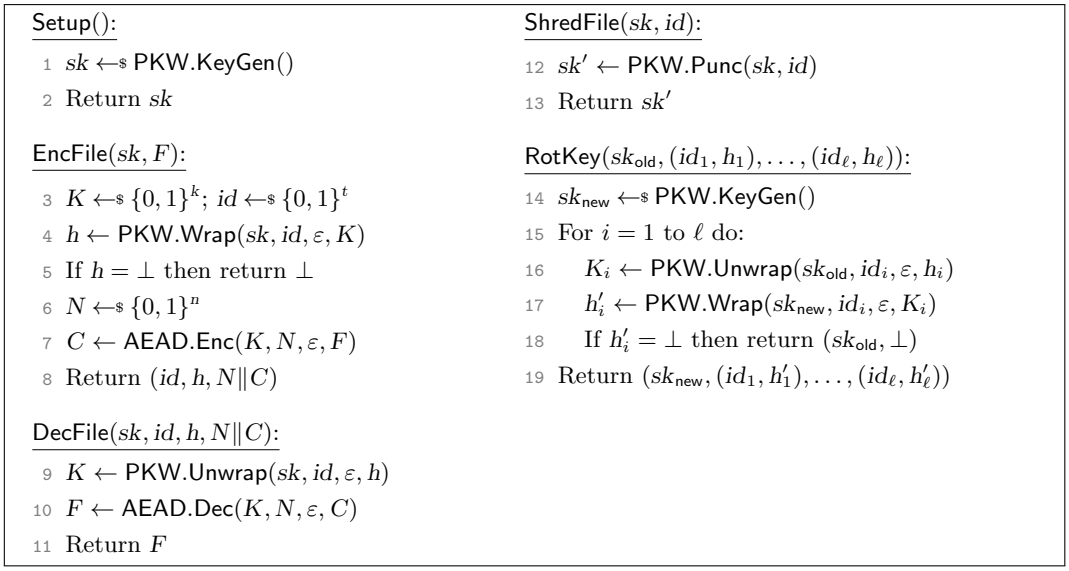

Figure 13: Construction of a protected file storage scheme PFS[PKW, AEAD] from a puncturable key-wrapping scheme PKW and an AEAD scheme AEAD. The PKW scheme has wrap-key space $\{0,1\}^k$ and tag space $\{0,1\}^t$. The AEAD scheme has key space $\{0,1\}^k$ and nonce space $\{0,1\}^n$. Hence, for the resulting PFS scheme, $\mathcal{I} = \{0,1\}^t$, $\mathcal{H} = \{0,1\}^{\mathsf{PKW.cl}(k)}$, and $\mathsf{PFS.cl}(|F|) = n + \mathsf{AEAD.cl}(|F|)$.

files by updating the key in a step-wise fashion (e.g. using ratcheting techniques), rather than by using the shredding algorithm of the PFS system.

6.3 Instantiating PFS from PKW and AEAD

We now construct a generic PFS scheme PFS[PKW, AEAD] from a puncturable key-wrapping scheme PKW, which will handle the key management, and an authenticated encryption scheme with associated data AEAD, handling the actual file encryption. The construction, formalized in Figure 13, works as follows.

Setup generates a PKW key sk, which—for reference to cloud storage and its key-wrapping functionality—we refer to as the key encryption key (KEK).

EncFile first samples an AEAD "data encryption key" (DEK) and a file identifier id at random, and wraps DEK under the KEK into a file header h, using id as tag.[7] It then AEAD-encrypts the file plaintext under DEK and a random[8] nonce N into a ciphertext C; $N\|C$ constitutes the PFS file ciphertext.

DecFile inverts file encryption by first unwrapping the DEK from the header and then using it to decrypt the file ciphertext.

ShredFile punctures the KEK sk on a file identifier id, using the PKW puncturing algorithm. This effectively prevents future unwrapping of the DEK wrapped with tag id, and hence file decryptions of files with this identifier.

RotKey first unwraps the DEKs in all headers it is handed, then samples a fresh KEK to re-wrap them. The PKW tags are re-used in this process, ensuring that encrypted files keep their identifiers across key rotations.

Security from puncturable key wrapping. The following theorems state that the construction meets the security goals for a protected file storage system. We establish confidentiality in Theorems 7

[7] Our construction leaves the PKW header empty. In practice, this field may be used to authenticate control data of the DEK, such as expiration date or permitted usage.

[8] Our construction only uses a single AEAD nonce N per any one data encryption key DEK, which would allow using a fixed nonce. We still sample a random nonce to enable file updates/re-encryption as a potential extension to our construction.

and 8 and integrity in Theorem 9. Notably, our two confidentiality results follow different paths: Theorem 7 employs weak one-time (find$-1cpa) PKW security in a hybrid together with puncture invariance and consistency to establish confidentiality for our PFS scheme. Theorem 8 in contrast shows our construction achieves the same goal in a tight manner if the underlying PKW scheme meets the stronger find$-rcpa notion. While the latter notion is currently only known to be achievable from strong (fpr-rro$) PPRF security, the route of Theorem 8 may still be interesting as it does not require puncture invariance and consistency, properties which we expect schemes with non-perfect correctness (e.g., employing Bloom filters), would not achieve. We state the theorems and provide proof sketches below. For the full proofs, see Appendix E.

Theorem 7 (PFS[PKW, AEAD] is find$-rcpa secure, via PKW find$-1cpa)**.** *Let* PFS[PKW, AEAD] *be the PFS construction in Figure 13 with file identifier space* $\mathcal{I} = \{0,1\}^t$*. If* PKW *is puncture invariant and consistent (Definitions 8 and 9), then for every adversary* $\mathcal{A}$ *against the* find$-rcpa *security (Definition 13) of* PFS[PKW, AEAD] *making at most* $q_{ro\$}$*,* q_e*, resp.* $m-1$ *queries in total to its oracles* RO$-ENC*,* ENC*, and* ROTKEY*, and at most* q_s *queries to oracle* SHRED *between each query to the key rotation oracle* ROTKEY*, there exists adversaries* $\mathcal{B}_{\mathsf{pkw}}$ *and* $\mathcal{B}_{\mathsf{aead}}$ *running in approximately the same time as* $\mathcal{A}$ *such that*

$$\mathbf{Adv}^{\text{find\$-rcpa}}_{\mathsf{PFS[PKW,AEAD]}}(\mathcal{A}) \leq 2q_{ro\$}\left(\frac{(2q_s+q_e+q_{ro\$}-1)}{2^t} + m\cdot\mathbf{Adv}^{\text{find\$-1cpa}}_{\mathsf{PKW}}(\mathcal{B}_{\mathsf{pkw}}) + m\cdot\mathbf{Adv}^{\text{ind\$-cpa}}_{\mathsf{AEAD}}(\mathcal{B}_{\mathsf{aead}})\right).$$

Adversary $\mathcal{B}_{\mathsf{pkw}}$ *makes at most* m *queries to oracle* NEW-RO$-WRAP*. Adversary* $\mathcal{B}_{\mathsf{aead}}$ *makes one query each to its oracles* NEW *and* RO$*.*

Proof idea. (For full proof, see Appendix E.1.)The proof proceeds by a series of six game hops, starting with game $\mathbf{G}_0 = \mathbf{G}^{\text{find\$-rcpa}}_{\mathsf{PFS[PKW,AEAD]}}$. Let $\mathbf{Adv}_i(\mathcal{A}) := 2\left|\Pr\left[\,\mathbf{G}_i(\mathcal{A})\,\right] - \frac{1}{2}\right|$ for $i \in \{0,\ldots,6\}$. By *key phase* we denote the period between two consecutive key rotation queries.

$\mathbf{G}_0 \to \mathbf{G}_1$: We begin by excluding, via a bad event [7], that the (real- or ideal-world) challenge file identifier coincides with one already shredded in the current key phase, since the output of wrapping with such an identifier as tag is undefined and hence possibly distinguishable from the ideal-world behavior. The probability of this happening is upper-bounded by $2q_{ro\$}\cdot\frac{q_s}{2^t}$.

$\mathbf{G}_1 \to \mathbf{G}_2$: We reduce the $q_{ro\$}$ RO$-ENC challenge queries to a single one via a hybrid argument, yielding an adversary $\mathcal{A}'$ making a single query to RO$-ENC and at most $q_e + q_{ro\$} - 1$ queries to ENC, such that $\mathbf{Adv}_1(\mathcal{A}) = q_{ro\$}\cdot\mathbf{Adv}_2(\mathcal{A}')$.

$\mathbf{G}_2 \to \mathbf{G}_3$: Next, we exclude that PKW tags used for the (at most $q_e + q_{ro\$} - 1$) real encryption queries prior to the challenge query collide with the (single) challenge tag, a bad event occurring with probability at most $\frac{q_e+q_{ro\$}-1}{2^t}$.

$\mathbf{G}_3 \to \mathbf{G}_4$: The challenger now guesses in which of the at most m key phases the challenge encryption occurs; silencing the output otherwise loses a factor of m.

$\mathbf{G}_4 \to \mathbf{G}_5$: We can now apply the find$-1cpa security of PKW through a reduction $\mathcal{B}_{\mathsf{pkw}}$ to replace the header in the challenge encryption by a random string. This step requires PKW's puncture invariance and consistency to reorder the challenge PKW wrap in the reduction; the latter makes at most m queries to oracle NEW-RO$-WRAP and yields $|\Pr\left[\,\mathbf{G}_4\,\right] - \Pr\left[\,\mathbf{G}_5\,\right]| \leq \mathbf{Adv}^{\text{find\$-1cpa}}_{\mathsf{PKW}}(\mathcal{B}_{\mathsf{pkw}})$.

$\mathbf{G}_5 \to \mathbf{G}_6$: Finally, we replace the challenge file ciphertext with a random string via a reduction $\mathcal{B}_{\mathsf{aead}}$ to the AEAD scheme's ind$-cpa security, which yields $|\Pr\left[\,\mathbf{G}_5\,\right] - \Pr\left[\,\mathbf{G}_6\,\right]| \leq \mathbf{Adv}^{\text{ind\$-cpa}}_{\mathsf{AEAD}}(\mathcal{B}_{\mathsf{aead}})$. After this step, $\mathbf{Adv}_6(\mathcal{A}) = 0$. □

Theorem 8 (PFS[PKW, AEAD] is find$-rcpa secure, via PKW find$-rcpa)**.** *Let* PFS[PKW, AEAD] *be the PFS construction in Figure 13 with file identifier space* $\mathcal{I} = \{0,1\}^t$*. For every adversary* $\mathcal{A}$ *against the* find$-rcpa *security (Definition 13) of* PFS[PKW, AEAD] *making at most* $q_{ro\$}$*,* q_e*,* q_{corr}*, resp.* q_{rk} *queries in total to its oracles* RO$-ENC*,* ENC*,* CORR *and* ROTKEY*, and at most* q_s *queries to oracle* SHRED *between each query to oracle* ROTKEY*, there exists adversaries* $\mathcal{B}_{\mathsf{pkw}}$ *and* $\mathcal{B}_{\mathsf{aead}}$ *running in approximately the same time as* $\mathcal{A}$ *such that*

$$\mathbf{Adv}^{\text{find\$-rcpa}}_{\mathsf{PFS[PKW,AEAD]}}(\mathcal{A}) \leq 2\cdot\left(\frac{2q_{ro\$}q_s}{2^t} + \frac{(q_e+q_{ro\$})^2}{2^{t+1}} + \mathbf{Adv}^{\text{find\$-rcpa}}_{\mathsf{PKW}}(\mathcal{B}_{\mathsf{pkw}}) + \mathbf{Adv}^{\text{ind\$-cpa}}_{\mathsf{AEAD}}(\mathcal{B}_{\mathsf{aead}})\right).$$

Adversary $\mathcal{B}_{\mathsf{pkw}}$ *makes at most* $q_{rk}+1$, $q_{ro\$}(q_{rk}+1)$, $q_e(q_{rk}+1)$, q_{corr} *and* $q_{rk}\cdot q_s$ *queries to oracles* New, Ro$-Wrap, Wrap, Corr *and* Punc, *respectively. Adversary* $\mathcal{B}_{\mathsf{aead}}$ *makes at most* $q_{ro\$}$ *queries each to its oracles* New *and* Ro$.

Proof idea. (For full proof, see Appendix E.2.) The proof does four game hops, starting from $\mathbf{G}_0 = \mathbf{G}^{\text{find\$-rcpa}}_{\mathsf{PFS[PKW,AEAD]}}$, using similar notation as in the proof of Theorem 7. The core difference to Theorem 7 is that the find$-rcpa security of the PKW scheme allows us to directly simulate the many challenge (and interleaved encryption) queries, without the need for reordering and a hybrid-argument reduction to a single challenge.

$\mathbf{G}_0 \to \mathbf{G}_1$: We first exclude real- and ideal-world challenge file identifiers colliding with shredded ones, yielding the term $\frac{2q_{ro\$}q_s}{2^t}$ as in the proof of Theorem 7.

$\mathbf{G}_1 \to \mathbf{G}_2$: We next exclude any collisions of sampled file identifiers among (real and challenge) encryption queries. By the birthday bound, such collisions happen with probability at most $\frac{(q_e+q_{ro\$})^2}{2^{t+1}}$.

$\mathbf{G}_2 \to \mathbf{G}_3$: We can now replace challenge headers by random strings through a reduction $\mathcal{B}_{\mathsf{pkw}}$ to the find$-rcpa security of PKW which makes at most $q_{ro\$}+q_{ro\$}\cdot q_{rk}$, resp. $q_e+q_e\cdot q_{rk}$ queries to oracle Ro$-Wrap resp. Wrap to simulate oracles Ro$-Enc and RotKey, resp. Enc and RotKey. This yields $|\Pr[\mathbf{G}_2]-\Pr[\mathbf{G}_3]| \le \mathbf{Adv}^{\text{find\$-rcpa}}_{\mathsf{PKW}}(\mathcal{B}_{\mathsf{pkw}})$.

$\mathbf{G}_3 \to \mathbf{G}_4$: Finally, we can replace file ciphertexts with random strings as in the proof of Theorem 7, completing the bound with $|\Pr[\mathbf{G}_3]-\Pr[\mathbf{G}_4]| \le \mathbf{Adv}^{\text{ind\$-cpa}}_{\mathsf{AEAD}}(\mathcal{B}_{\mathsf{aead}})$, as afterwards $\mathbf{Adv}_4(\mathcal{A})=0$. □

Theorem 9 (PFS[PKW, AEAD] is int-ctxt secure)**.** *Let* PFS[PKW, AEAD] *be the PFS construction in Figure 13 with file identifier space* $\mathcal{I}=\{0,1\}^t$. *For every adversary* $\mathcal{A}$ *against the* int-ctxt *security of* PFS[PKW, AEAD], *making at most* q_e *queries to oracle* Enc, *there exist adversaries* $\mathcal{B}_{\mathsf{pkw}}$, $\mathcal{C}_{\mathsf{pkw}}$ *and* $\mathcal{B}_{\mathsf{aead}}$ *running in approximately the same time as* $\mathcal{A}$ *such that*

$$\mathbf{Adv}^{\text{int-ctxt}}_{\mathsf{PFS[PKW,AEAD]}}(\mathcal{A}) \le \frac{q_e^2}{2^{t+1}} + \mathbf{Adv}^{\text{int-ctxt}}_{\mathsf{PKW}}(\mathcal{B}_{\mathsf{pkw}}) + \mathbf{Adv}^{\text{find\$-cpa}}_{\mathsf{PKW}}(\mathcal{C}_{\mathsf{pkw}}) + \mathbf{Adv}^{\text{int-ctxt}}_{\mathsf{AEAD}}(\mathcal{B}_{\mathsf{aead}}),$$

where t *is the bit length of the file identifiers.*

Let q_e, q_d, q_s *and* q_{rk} *be the maximum number of queries by adversary* $\mathcal{A}$ *to oracle* Enc, Dec, Shred *and* RotKey, *respectively. Furthermore let* h *be the maximum number of file identifier and header pairs in any one query to oracle* RotKey. *Then adversary* $\mathcal{B}_{\mathsf{pkw}}$ *makes at most* $q_{rk}+1$ *queries to oracle* New, $q_e+h\cdot q_{rk}$ *queries to oracle* Wrap, $q_d+h\cdot q_{rk}$ *queries to oracle* Unwrap *and* q_s *queries to oracle* Punc *in game* $\mathbf{G}^{\text{int-ctxt}}_{\mathsf{PKW}}$. *Adversary* $\mathcal{C}_{\mathsf{pkw}}$ *makes at most* $q_{rk}+1$ *queries to oracle* New, $q_e+h\cdot q_{rk}$ *queries to oracle* Ro$-Wrap *and* q_s *queries to oracle* Punc *in game* $\mathbf{G}^{\text{find\$-cpa}}_{\mathsf{PKW}}$. *Adversary* $\mathcal{B}_{\mathsf{aead}}$ *makes at most* q_e *calls to oracle* New, q_e *calls to oracle* Enc *and* q_d *calls to oracle* Dec *in game* $\mathbf{G}^{\text{int-ctxt}}_{\mathsf{AEAD}}$.

Proof idea. (For full proof, see Appendix E.3.) The proof has three game hops, starting from $\mathbf{G}_0 = \mathbf{G}^{\text{int-ctxt}}_{\mathsf{PFS[PKW,AEAD]}}$.

$\mathbf{G}_0 \to \mathbf{G}_1$: First, we exclude tag collisions in the underlying PKW scheme to remove the need to handle encryption queries with coinciding file identifiers. The probability of such is upper bounded via the birthday bound by $\frac{q_e^2}{2^{t+1}}$.

$\mathbf{G}_1 \to \mathbf{G}_2$: Next, we apply integrity of ciphertexts of the PKW scheme to ensure that in any tuple (id, h, C) which successfully decrypts, the file identifier and header (id, h) were honestly generated by the wrap algorithm of the PKW scheme in a previous encryption query. Since (id, h) fully determines the data encryption key[9], this means that all forgery attempts occur under an AEAD key which has been wrapped by the PKW scheme during a preceding encryption query. Via a reduction $\mathcal{B}_{\mathsf{pkw}}$, this step is bounded by $\mathbf{Adv}^{\text{int-ctxt}}_{\mathsf{PKW}}(\mathcal{B}_{\mathsf{pkw}})$.

$\mathbf{G}_2 \to \mathbf{G}_3$: In the last game hop we reduce to PKW find$-cpa security to replace all headers by random strings, introducing the bound $\mathbf{Adv}^{\text{find\$-cpa}}_{\mathsf{PKW}}(\mathcal{C}_{\mathsf{pkw}})$. The DEKs used for file encryption and decryption are

[9] The construction PFS[PKW, AEAD] leaves the PKW header field empty, hence (id, h) determines the wrapped K via the determinism of PKW.Unwrap.

thus independent from the headers, and by construction they are sampled u.a.r. by the file encryption algorithm.

The adversary's advantage in game $\mathbf{G}_3$ is finally bounded by the advantage $\mathbf{Adv}^{\text{int-ctxt}}_{\mathsf{AEAD}}(\mathcal{B}_{\mathsf{aead}})$ of a reduction to the multi-key integrity of AEAD. □

7 Discussion and Future Work

Our approach to PKW integrates a flexible tag-based approach [35] with classical key wrapping [51]. We build PKW generically from PPRF and AEAD, focusing on applications which require fine-grained forward security. For applications where batch puncturing might be useful, deploying nonce-misuse resistant AEAD would allow tags to be reused, achieving a stronger version of our main find$-cpa security notion. Interestingly, proving the (even stronger) find$-rcpa security of such an instantiation runs into a key commitment problem; whether resolving this needs idealized models (cf. [39]) or can be done in the standard model is an interesting open problem.

We introduced several new definitions for PPRFs, motivated by their use in constructing PKW schemes and realizing our target applications. Making a full investigation of how these notions relate to existing definitions, and how they can be efficiently realized, would be of foundational interest.

Our PKWs and the PPRFs they are built from are not private [11]; we could potentially obtain improved privacy after client compromise for our PFS system if they were, cf. [58]. Finding practically efficient private PPRFs and building private PKW schemes from them is an open problem whose solution would have immediate applications.

Our work on TLS session resumption assumes the server's key is held and operated on by a single server. Yet distributed server environments are common in TLS deployments, to reduce latency and improve scalability. It would be useful to extend our work to this setting. The challenge is to maintain appropriate synchronization amongst the punctured keys held by the servers.

The applications we treat in this work are a sample from the set of possible use-cases for PKW. They already demonstrate that it is a useful abstraction. Examining further potential applications where puncturable key wrapping can be integrated, such as in symmetric key exchange [15] and DUKPT [18], would be interesting future work.

Acknowledgments

We thank the anonymous reviewers for their helpful comments. Felix Günther has been supported in part by Research Fellowship grant GU 1859/1-1 of the German Research Foundation (DFG). A big thank you to Kien Tuong Truong for the key rap in Appendix F.

References

[1] Nimrod Aviram, Kai Gellert, and Tibor Jager. Session resumption protocols and efficient forward security for TLS 1.3 0-RTT. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part II*, volume 11477 of *LNCS*, pages 117–150. Springer, Heidelberg, May 2019.

[2] Nimrod Aviram, Kai Gellert, and Tibor Jager. Session resumption protocols and efficient forward security for TLS 1.3 0-RTT. *Journal of Cryptology*, 34(3):20, July 2021.

[3] Gildas Avoine, Sébastien Canard, and Loïc Ferreira. Symmetric-key authenticated key exchange (SAKE) with perfect forward secrecy. In Stanislaw Jarecki, editor, *CT-RSA 2020*, volume 12006 of *LNCS*, pages 199–224. Springer, Heidelberg, February 2020.

[4] AWS. Protecting data using client-side encryption. http://docs.aws.amazon.com/AmazonS3/latest/dev/UsingClientSideEncryption.html.

[5] Mihir Bellare, Alexandra Boldyreva, and Silvio Micali. Public-key encryption in a multi-user setting: Security proofs and improvements. In Bart Preneel, editor, *EUROCRYPT 2000*, volume 1807 of *LNCS*, pages 259–274. Springer, Heidelberg, May 2000.

[6] Mihir Bellare, Ruth Ng, and Björn Tackmann. Nonces are noticed: AEAD revisited. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part I*, volume 11692 of *LNCS*, pages 235–265. Springer, Heidelberg, August 2019.

[7] Mihir Bellare and Phillip Rogaway. The security of triple encryption and a framework for code-based game-playing proofs. In Serge Vaudenay, editor, *EUROCRYPT 2006*, volume 4004 of *LNCS*, pages 409–426. Springer, Heidelberg, May / June 2006.

[8] Mihir Bellare, Igors Stepanovs, and Brent Waters. New negative results on differing-inputs obfuscation. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part II*, volume 9666 of *LNCS*, pages 792–821. Springer, Heidelberg, May 2016.

[9] Matt Blaze. A cryptographic file system for UNIX. In Dorothy E. Denning, Raymond Pyle, Ravi Ganesan, Ravi S. Sandhu, and Victoria Ashby, editors, *ACM CCS 93*, pages 9–16. ACM Press, November 1993.

[10] Dan Boneh, Sam Kim, and Hart William Montgomery. Private puncturable PRFs from standard lattice assumptions. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *EUROCRYPT 2017, Part I*, volume 10210 of *LNCS*, pages 415–445. Springer, Heidelberg, April / May 2017.

[11] Dan Boneh, Kevin Lewi, and David J. Wu. Constraining pseudorandom functions privately. In Serge Fehr, editor, *PKC 2017, Part II*, volume 10175 of *LNCS*, pages 494–524. Springer, Heidelberg, March 2017.

[12] Dan Boneh and Richard J. Lipton. A revocable backup system. In *USENIX Security 96*. USENIX Association, July 1996.

[13] Dan Boneh and Brent Waters. Constrained pseudorandom functions and their applications. In Kazue Sako and Palash Sarkar, editors, *ASIACRYPT 2013, Part II*, volume 8270 of *LNCS*, pages 280–300. Springer, Heidelberg, December 2013.

[14] Boxcryptor. Boxcryptor security for your cloud. https://www.boxcryptor.com/.

[15] Colin Boyd, Gareth T. Davies, Bor de Kock, Kai Gellert, Tibor Jager, and Lise Millerjord. Symmetric key exchange with full forward security and robust synchronization. In Mehdi Tibouchi and Huaxiong Wang, editors, *ASIACRYPT 2021, Part IV*, volume 13093 of *LNCS*, pages 681–710. Springer, Heidelberg, December 2021.

[16] Colin Boyd and Kai Gellert. A modern view on forward security. *Comput. J.*, 64(4):639–652, 2021.

[17] Elette Boyle, Shafi Goldwasser, and Ioana Ivan. Functional signatures and pseudorandom functions. In Hugo Krawczyk, editor, *PKC 2014*, volume 8383 of *LNCS*, pages 501–519. Springer, Heidelberg, March 2014.

[18] Eric Brier and Thomas Peyrin. A forward-secure symmetric-key derivation protocol - how to improve classical DUKPT. In Masayuki Abe, editor, *ASIACRYPT 2010*, volume 6477 of *LNCS*, pages 250–267. Springer, Heidelberg, December 2010.

[19] Ran Canetti and Yilei Chen. Constraint-hiding constrained PRFs for NC^1 from LWE. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *EUROCRYPT 2017, Part I*, volume 10210 of *LNCS*, pages 446–476. Springer, Heidelberg, April / May 2017.

[20] Weikeng Chen, Thang Hoang, Jorge Guajardo, and Attila A. Yavuz. Titanium: A metadata-hiding file-sharing system with malicious security. In *NDSS 2022*. The Internet Society, 2022.

[21] Weikeng Chen and Raluca Ada Popa. Metal: A metadata-hiding file-sharing system. In *NDSS 2020*. The Internet Society, February 2020.

[22] Katriel Cohn-Gordon, Cas J. F. Cremers, and Luke Garratt. On post-compromise security. In Michael Hicks and Boris Köpf, editors, *CSF 2016 Computer Security Foundations Symposium*, pages 164–178. IEEE Computer Society Press, 2016.

[23] David Derler, Tibor Jager, Daniel Slamanig, and Christoph Striecks. Bloom filter encryption and applications to efficient forward-secret 0-RTT key exchange. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part III*, volume 10822 of *LNCS*, pages 425–455. Springer, Heidelberg, April / May 2018.

[24] Whitfield Diffie, Paul C. van Oorschot, and Michael J. Wiener. Authentication and authenticated key exchanges. *Designs, Codes and Cryptography*, 2(2):107–125, June 1992.

[25] Benjamin Dowling, Marc Fischlin, Felix Günther, and Douglas Stebila. A cryptographic analysis of the TLS 1.3 handshake protocol candidates. In Indrajit Ray, Ninghui Li, and Christopher Kruegel, editors, *ACM CCS 2015*, pages 1197–1210. ACM Press, October 2015.

[26] Benjamin Dowling, Marc Fischlin, Felix Günther, and Douglas Stebila. A cryptographic analysis of the TLS 1.3 handshake protocol. *Journal of Cryptology*, 34(4):37, October 2021.

[27] Morris Dworkin. Recommendation for block cipher modes of operation: Methods for key wrapping. NIST Special Publication SP 800-38F, 2012.

[28] Adam Everspaugh, Kenneth G. Paterson, Thomas Ristenpart, and Samuel Scott. Key rotation for authenticated encryption. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part III*, volume 10403 of *LNCS*, pages 98–129. Springer, Heidelberg, August 2017.

[29] Ariel J. Feldman, William P. Zeller, Michael J. Freedman, and Edward W. Felten. Sporc: Group collaboration using untrusted cloud resources. In *OSDI 20210*, 2010.

[30] Marc Fischlin and Felix Günther. Multi-stage key exchange and the case of Google's QUIC protocol. In Gail-Joon Ahn, Moti Yung, and Ninghui Li, editors, *ACM CCS 2014*, pages 1193–1204. ACM Press, November 2014.

[31] Marc Fischlin and Felix Günther. Replay attacks on zero round-trip time: The case of the TLS 1.3 handshake candidates. In *EuroS&P 2017*, pages 60–75. IEEE, April 2017.

[32] Georg Fuchsbauer, Momchil Konstantinov, Krzysztof Pietrzak, and Vanishree Rao. Adaptive security of constrained PRFs. In Palash Sarkar and Tetsu Iwata, editors, *ASIACRYPT 2014, Part II*, volume 8874 of *LNCS*, pages 82–101. Springer, Heidelberg, December 2014.

[33] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions (extended abstract). In *25th FOCS*, pages 464–479. IEEE Computer Society Press, October 1984.

[34] Google. Encryption at rest in Google Cloud. https://cloud.google.com/security/encryption/default-encryption.

[35] Matthew D. Green and Ian Miers. Forward secure asynchronous messaging from puncturable encryption. In *2015 IEEE Symposium on Security and Privacy*, pages 305–320. IEEE Computer Society Press, May 2015.

[36] Christoph G. Günther. An identity-based key-exchange protocol. In Jean-Jacques Quisquater and Joos Vandewalle, editors, *EUROCRYPT'89*, volume 434 of *LNCS*, pages 29–37. Springer, Heidelberg, April 1990.

[37] Felix Günther, Britta Hale, Tibor Jager, and Sebastian Lauer. 0-RTT key exchange with full forward secrecy. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *EUROCRYPT 2017, Part III*, volume 10212 of *LNCS*, pages 519–548. Springer, Heidelberg, April / May 2017.

[38] IBM. Protecting data with envelope encryption. https://cloud.ibm.com/docs/key-protect?topic=key-protect-envelope-encryption.

[39] Joseph Jaeger and Nirvan Tyagi. Handling adaptive compromise for practical encryption schemes. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part I*, volume 12170 of *LNCS*, pages 3–32. Springer, Heidelberg, August 2020.

[40] Aggelos Kiayias, Stavros Papadopoulos, Nikos Triandopoulos, and Thomas Zacharias. Delegatable pseudorandom functions and applications. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *ACM CCS 2013*, pages 669–684. ACM Press, November 2013.

[41] Michael Klooß, Anja Lehmann, and Andy Rupp. (R)CCA secure updatable encryption with integrity protection. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part I*, volume 11476 of *LNCS*, pages 68–99. Springer, Heidelberg, May 2019.

[42] Hugo Krawczyk. Cryptographic extraction and key derivation: The HKDF scheme. In Tal Rabin, editor, *CRYPTO 2010*, volume 6223 of *LNCS*, pages 631–648. Springer, Heidelberg, August 2010.

[43] Billy Lau, Simon P. Chung, Chengyu Song, Yeongjin Jang, Wenke Lee, and Alexandra Boldyreva. Mimesis aegis: A mimicry privacy shield-A system's approach to data privacy on public cloud. In Kevin Fu and Jaeyeon Jung, editors, *USENIX Security 2014*, pages 33–48. USENIX Association, August 2014.

[44] Anja Lehmann and Björn Tackmann. Updatable encryption with post-compromise security. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part III*, volume 10822 of *LNCS*, pages 685–716. Springer, Heidelberg, April / May 2018.

[45] Prince Mahajan, Srinath Setty, Sangmin Lee, Allen Clement, Lorenzo Alvisi, Mike Dahlin, and Michael Walfish. Depot: Cloud storage with minimal trust. *ACM Trans. Comput. Syst.*, 29(4), dec 2011.

[46] Microsoft. Azure Data Encryption at rest. https://docs.microsoft.com/en-us/azure/security/fundamentals/encryption-atrest.

[47] E. Miller, D. Long, W. Freeman, and B. Reed. Strong security for distributed file systems. In *Conference Proceedings of the 2001 IEEE International Performance, Computing, and Communications Conference*, pages 34–40, 2001.

[48] Shaun Nichols. Dropbox: Oops, yeah, we didn't actually delete all your files this bug kept them in the cloud. https://www.theregister.com/2017/01/24/dropbox_brings_old_files_back_from_dead/, 2017.

[49] E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.3. RFC 8446 (Proposed Standard), August 2018.

[50] Phillip Rogaway. Authenticated-encryption with associated-data. In Vijayalakshmi Atluri, editor, *ACM CCS 2002*, pages 98–107. ACM Press, November 2002.

[51] Phillip Rogaway and Thomas Shrimpton. A provable-security treatment of the key-wrap problem. In Serge Vaudenay, editor, *EUROCRYPT 2006*, volume 4004 of *LNCS*, pages 373–390. Springer, Heidelberg, May / June 2006.

[52] Amit Sahai and Brent Waters. How to use indistinguishability obfuscation: deniable encryption, and more. In David B. Shmoys, editor, *46th ACM STOC*, pages 475–484. ACM Press, May / June 2014.

[53] J. Salowey, H. Zhou, P. Eronen, and H. Tschofenig. Transport Layer Security (TLS) Session Resumption without Server-Side State. RFC 5077 (Proposed Standard), January 2008. Obsoleted by RFC 8446, updated by RFC 8447.

[54] Daniel Slamanig and Christoph Striecks. Puncture 'em all: Updatable encryption with no-directional key updates and expiring ciphertexts. Cryptology ePrint Archive, Report 2021/268, 2021. https://eprint.iacr.org/2021/268.

[55] Shi-Feng Sun, Ron Steinfeld, Shangqi Lai, Xingliang Yuan, Amin Sakzad, Joseph K. Liu, Surya Nepal, and Dawu Gu. Practical non-interactive searchable encryption with forward and backward privacy. In *NDSS 2021*. The Internet Society, February 2021.

[56] Shifeng Sun, Xingliang Yuan, Joseph K. Liu, Ron Steinfeld, Amin Sakzad, Viet Vo, and Surya Nepal. Practical backward-secure searchable encryption from symmetric puncturable encryption. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *ACM CCS 2018*, pages 763–780. ACM Press, October 2018.

[57] Erik Sy, Christian Burkert, Hannes Federrath, and Mathias Fischer. Tracking users across the web via TLS session resumption. In *ACSAC 2018*, pages 289–299. ACM, 2018.

[58] Nirvan Tyagi, Muhammad Haris Mughees, Thomas Ristenpart, and Ian Miers. BurnBox: Self-revocable encryption in a world of compelled access. In William Enck and Adrienne Porter Felt, editors, *USENIX Security 2018*, pages 445–461. USENIX Association, August 2018.

A PPRF Relations

Theorem 10 (PPRF: fpr-1ro$ $\implies$ fpr-ro$). *Let* $\mathsf{PPRF} = (\mathsf{KeyGen}, \mathsf{Eval}, \mathsf{Punc})$ *be a PPRF scheme. If* PPRF *is* puncture invariant *(Definition 5), then for every adversary* $\mathcal{A}$ *against the* fpr-ro$ *security of* PPRF *making at most* $q_{ro\$}$ *distinct queries to oracle* RO$-EVAL *under each key, and at most* q_n *to oracle* NEW*, there exists an adversary* $\mathcal{B}$ *running in approximately the same time as* $\mathcal{A}$*, such that*

$$\mathbf{Adv}^{\text{fpr-ro\$}}_{\mathsf{PPRF}}(\mathcal{A}) \leq q_{ro\$} \cdot \mathbf{Adv}^{\text{fpr-1ro\$}}_{\mathsf{PPRF}}(\mathcal{B}).$$

Adversary $\mathcal{B}$ *makes at most* q_n *queries to oracle* NEW-RO$-EVAL.

Proof. The proof consists of a standard hybrid argument across $q_{ro\$}+1$ games $\mathbf{H}_0$-$\mathbf{H}_{q_{ro\$}}$, where game $\mathbf{H}_j$ works as $\mathbf{G}^{\text{fpr-ro\$}}_{\mathsf{PPRF}}$, except that instead of answering all RO$-EVAL queries with real evaluations or random strings depending on a secret bit, in game $\mathbf{H}_j$ the first j queries under each key are answered with strings drawn u.a.r. in the range and the remaining queries are answered with real PPRF evaluations. Adversary $\mathcal{B}$ simulates game $\mathbf{H}_j$ for $\mathcal{A}$ by storing any puncturing queries prior to the j^{th} RO$-EVAL query for each key, and after relaying the j^{th} query to oracle NEW-RO$-EVAL, puncturing the key it receives in return on the stored points. Puncture invariance ensures that the potential reordering of punctures (due to the automatic puncturing by oracle NEW-RO$-EVAL on the challenge point) does not affect the distribution of the secret key and hence that corruption queries can be perfectly simulated. We also rely on PPRF correctness to ensure that the response to the RO$-EVAL query is correctly distributed. When $\mathcal{A}$ halts and outputs a bit b^*, adversary $\mathcal{B}$ also halts and outputs b^*. This way, adversary $\mathcal{B}$ perfectly simulates the hybrid games for $\mathcal{A}$ and achieves the claimed advantage bound. □

Theorem 11 (PPRF: fpr-ro$ $\implies$ fpr-rro$). *Let* $\mathsf{PPRF} = (\mathsf{KeyGen}, \mathsf{Eval}, \mathsf{Punc})$ *be a PPRF scheme with domain* $\mathcal{X}$ *and range* $\mathcal{Y}$*. If* PPRF *is* puncture invariant *(Definition 5), then for every adversary* $\mathcal{A}$ *against the* fpr-rro$ *security of* PPRF *making at most* $q_{ro\$}$ *distinct queries to oracle* RO$-EVAL *under each key, and at most* q_n *to oracle* NEW*, there exists an adversary* $\mathcal{B}$ *running in approximately the same time as* $\mathcal{A}$*, such that*

$$\mathbf{Adv}^{\text{fpr-rro\$}}_{\mathsf{PPRF}}(\mathcal{A}) \leq q_{ro\$} q_n \cdot |\mathcal{X}| \cdot \mathbf{Adv}^{\text{fpr-ro\$}}_{\mathsf{PPRF}}(\mathcal{B}).$$

Adversary $\mathcal{B}$ *makes only a single query each to oracles* NEW *and* RO$-EVAL.

Proof. First, we reduce the multi-key fpr-rro$ security to a single-key version via a standard hybrid argument. Let $\mathbf{G}_0$ be the original fpr-rro$ security game, and let $\mathbf{G}_1$ be identical to $\mathbf{G}_0$, except that there is no NEW-oracle. Instead the (single) key is drawn by the challenger at the start of the game, and all oracle queries are answered with respect to this key. We construct an adversary $\mathcal{B}_1$ such that

$$\mathbf{Adv}_0(\mathcal{A}) \leq q_n \cdot \mathbf{Adv}_1(\mathcal{B}_1), \tag{1}$$

where going forward we let $\mathbf{Adv}_i(\mathcal{D}) := 2\left|\Pr\left[\,\mathbf{G}_i(\mathcal{D})\,\right] - \frac{1}{2}\right|$ for any adversary $\mathcal{D}$ and $i \in \mathbb{N}$. The hybrid works across q_n+1 games $\mathbf{H}_0$-$\mathbf{H}_{q_n}$, where game $\mathbf{H}_j$ is identical to $\mathbf{G}^{\text{fpr-rro\$}}_{\mathsf{PPRF}}$, except that instead of picking a random bit b which determines if RO$-EVAL-queries are answered with real or random responses, in game $\mathbf{H}_j$ all queries made to the first j keys are answered with real PPRF evaluations and the queries under the rest of the q_n keys are answered with a string drawn u.a.r. from $\mathcal{Y}$. Adversary $\mathcal{B}_1$ samples an index ℓ in $\{1, \ldots, q_{ro\$}\}$ and simulates game $\mathbf{H}_\ell$ for $\mathcal{A}$, acting as the challenger, except that it forwards queries under key j to the oracles in game $\mathbf{G}_1$. When adversary $\mathcal{A}$ halts and outputs a bit b^*, so does $\mathcal{B}_1$. This gives Equation (1).

Next, via another hybrid argument, we hop to game $\mathbf{G}_2$ in which the adversary can only make a single challenge query to oracle RO$-EVAL. The argument uses a series of $q_{ro\$}+1$ games $\mathbf{H}'_0$-$\mathbf{H}'_{q_{ro\$}}$. Here game $\mathbf{H}'_j$ is identical to $\mathbf{G}_1$, except that instead of answering all RO$-EVAL queries with real evaluations or random strings depending on a secret bit, in game $\mathbf{H}_j$ the first j queries are answered with strings drawn u.a.r. in the range and the remaining queries are answered with real PPRF evaluations. We construct an adversary $\mathcal{B}_2$ which draws an index ℓ u.a.r. from $\{1, \ldots, q_{ro\$}\}$ and acts as the challenger in game $\mathbf{H}'_\ell$, except that it uses its RO$-EVAL to respond to the ℓ^{th} RO$-EVAL query from $\mathcal{B}_1$. When $\mathcal{B}_1$ halts and outputs b^*, then $\mathcal{B}_2$ halts and returns b^* as well. This gives

$$\mathbf{Adv}_1(\mathcal{B}_1) \leq q_{ro\$} \cdot \mathbf{Adv}_2(\mathcal{B}_2). \tag{2}$$

Lastly, we construct an adversary $\mathcal{B}$ against the fpr-rro$ security of PPRF such that

$$\mathbf{Adv}_2(\mathcal{B}_2) \leq |\mathcal{X}| \cdot \mathbf{Adv}^{\text{fpr-ro\$}}_{\mathsf{PPRF}}(\mathcal{B}). \tag{3}$$

The reduction relies on guessing the adaptive RO$-EVAL-query of adversary $\mathcal{B}_2$ in advance. Let b denote the hidden bit drawn by the challenger in game $\mathbf{G}_2$. We construct adversary $\mathcal{B}$ as follows. Adversary $\mathcal{B}$ begins by querying oracle NEW to generate a secret key for the game. It then draws a challenge point $\tilde{x}$ u.a.r. from the PPRF domain $\mathcal{X}$ and queries oracle RO$-EVAL on $\tilde{x}$, followed by a PUNC-query on the same point, after which it corrupts the secret key. Given that PPRF is puncture invariant, this enables $\mathcal{B}$ to simulate the responses to any EVAL-queries from $\mathcal{B}_2$ using the corrupted secret key, except on $\tilde{x}$. If adversary $\mathcal{B}_2$ queries $\tilde{x}$ to the simulated EVAL-oracle, then adversary $\mathcal{B}$ halts and returns a random bit b' as its output. Otherwise it continues to simulate game $\mathbf{G}_2$ until adversary $\mathcal{B}_2$ halts and returns its RO$-EVAL challenge point x^*. If $x^* \neq \tilde{x}$, then adversary $\mathcal{B}$ halts and outputs a random bit b'. If $x^* = \tilde{x}$, then adversary $\mathcal{B}$ returns the response it received from the RO$-EVAL on $\tilde{x}$ and continues to simulate game $\mathbf{G}_2$. When adversary $\mathcal{B}_2$ halts and returns bit b^*, adversary $\mathcal{B}$ also halts and returns $b' := b^*$.

We analyze the advantage of adversary $\mathcal{B}$. Let E_1 denote the event that adversary $\mathcal{B}_2$ does not query $\tilde{x}$ to EVAL and let E_2 denote that both E_1 occurred and that $\mathcal{B}_2$ picks $x^* = \tilde{x}$ as its challenge query. Let q_e denote the number of distinct EVAL queries issued by $\mathcal{B}_2$. Then

$$\Pr\left[\,\mathsf{E}_2\,\right] = \Pr\left[\,\mathsf{E}_1\,\right] \cdot \Pr\left[\,x^* = \tilde{x} \,\middle|\, \mathsf{E}_1\,\right] = \left(1 - \frac{q_e}{|\mathcal{X}|}\right) \cdot \frac{1}{|\mathcal{X}| - q_e} = \frac{1}{|\mathcal{X}|},$$

since $\tilde{x}$ is sampled u.a.r. from $\mathcal{X}$ and is unknown to adversary $\mathcal{B}_2$ (and hence independent from any queries made by $\mathcal{B}_2$). Adversary $\mathcal{B}$'s simulation of game $\mathbf{G}_2$ is perfect conditioned on event E_2, so $\mathbf{Adv}_2(\mathcal{B}_2) = 2\left|\Pr\left[\,b^* = b \,\middle|\, \mathsf{E}_2\,\right] - \frac{1}{2}\right|$. Now, $\Pr\left[\,\mathbf{G}^{\text{fpr-ro\$}}_{\mathsf{PPRF}}(\mathcal{B}) \,\middle|\, \overline{\mathsf{E}_2}\,\right] = \Pr\left[\,b' = b \,\middle|\, \overline{\mathsf{E}_2}\,\right] = 1/2$ by definition of adversary $\mathcal{B}$, since it outputs a random bit as its guess in the complement of event E_2. Putting this together, we see that

$$\begin{aligned}
\mathbf{Adv}^{\text{fpr-ro\$}}_{\mathsf{PPRF}}(\mathcal{B}) &= 2\left|\Pr\left[\,b' = b \,\middle|\, \mathsf{E}_2\,\right] \cdot \Pr\left[\,\mathsf{E}_2\,\right] + \Pr\left[\,b' = b \,\middle|\, \overline{\mathsf{E}_2}\,\right] \cdot \Pr\left[\,\overline{\mathsf{E}_2}\,\right] - \frac{1}{2}\right| \\
&= 2\left|\Pr\left[\,b' = b \,\middle|\, \mathsf{E}_2\,\right] \cdot \frac{1}{|\mathcal{X}|} + \frac{1}{2} \cdot \frac{|\mathcal{X}| - 1}{|\mathcal{X}|} - \frac{1}{2}\right| \\
&= \frac{1}{|\mathcal{X}|} \cdot 2\left|\Pr\left[\,b' = b \,\middle|\, \mathsf{E}_2\,\right] - \frac{1}{2}\right| = \frac{1}{|\mathcal{X}|} \cdot \mathbf{Adv}_2(\mathcal{B}_2),
\end{aligned}$$

proving the claimed bound in Equation (3). Putting together Equations (1)-(3) gives the bound in the theorem statement. □

B PKW Relations

First, we show that our "one challenge" notion find$-1cpa is in general weaker than the multi-challenge notion find$-cpa. That is, there exists a PKW scheme which is find$-1cpa secure, but not find$-cpa secure.

Theorem 12 (PKW: find$-1cpa $\not\Longrightarrow$ find$-cpa)**.** *Assume there exists a* find$-1cpa *secure PKW scheme* PKW *with ciphertext length function* cl*. Then there exists a PKW scheme* PKW′ *with ciphertext function* cl *which is* find$-1cpa *secure, but not* find$-cpa *secure.*

Proof. We modify scheme PKW to let the secret key include a set of tags, initially empty, to which Punc adds any tag that was punctured on. We further let $\mathsf{Wrap}(sk, T, H, K)$ output $0^{\mathsf{cl}(K)}$ when attempting to wrap key K with a (so-remembered) punctured tag. Note that this is not a problem for correctness, since it only applies to unpunctured tags. Otherwise the new scheme PKW′ is identical to PKW. The construction is given in Figure 14.

The find$-1cpa security of PKW′ directly reduces to that of PKW, as the game never calls algorithm Wrap after Punc. To violate find$-cpa security, an adversary $\mathcal{A}$ can query PUNC on an arbitrary tag T, then RO$-WRAP using the same tag T and an arbitrary header H and key K. The latter will always output $0^{\mathsf{cl}(K)}$ if the hidden bit is 1, otherwise $C \neq 0^{\mathsf{cl}(K)}$ with overwhelming probability $(1 - \frac{1}{2^{\mathsf{cl}(|K|)}})$. Hence, $\mathbf{Adv}^{\text{find\$-cpa}}_{\mathsf{PKW}'}(\mathcal{A}) = 1 - \frac{1}{2^{\mathsf{cl}(|K|)}}$. □

Next, we show that under certain restrictions—namely that the PKW scheme is puncture invariant, consistent and outputs $\perp$ on punctured tags—find$-1cpa implies find$-cpa security.

$\underline{\mathsf{KeyGen}'()}$:

1 $sk \leftarrow \mathsf{PKW.KeyGen}()$

2 $\mathcal{P} \leftarrow \emptyset$; $sk' \leftarrow (sk, \mathcal{P})$

3 Return sk'

$\underline{\mathsf{Wrap}'(sk', T, H, K)}$:

4 $(sk, \mathcal{P}) \leftarrow sk'$

5 If $T \in \mathcal{P}$ then return $0^{\mathsf{cl}(|K|)}$

6 $C \leftarrow \mathsf{PKW.Wrap}(sk, T, H, K)$

7 Return C

$\underline{\mathsf{Unwrap}'(sk', T, H, C)}$:

8 $(sk, \mathcal{P}) \leftarrow sk'$

9 Return $\mathsf{PKW.Unwrap}(sk, T, H, C)$

$\underline{\mathsf{Punc}'(sk', T)}$:

10 $(sk, \mathcal{P}) \leftarrow sk'$

11 $sk \leftarrow \mathsf{PKW.Punc}(sk, T)$

12 $\mathcal{P} \xleftarrow{\cup} \{T\}$

13 Return $(sk, \mathcal{P})$

Figure 14: Construction of PKW scheme PKW' for Theorem 12. The construction is parameterized by $\mathsf{PKW} = (\mathsf{KeyGen}, \mathsf{Wrap}, \mathsf{Unwrap}, \mathsf{Punc})$ with ciphertext-length function cl.

Theorem 13 (PKW: find\$-1cpa $\implies$ find\$-cpa)**.** *Let* $\mathsf{PKW} = (\mathsf{KeyGen}, \mathsf{Wrap}, \mathsf{Unwrap}, \mathsf{Punc})$ *be a puncture-invariant and consistent PKW scheme, for which* $\mathsf{Wrap}(sk, T, \cdot, \cdot)$ *outputs* $\perp$ *if* sk *has been previously punctured on* T*. Then for every adversary* $\mathcal{A}$ *against the* find\$-cpa *security of* PKW *making at most* $q_{ro\$}$ *distinct queries to oracle* RO\$-WRAP *under each key, and at most* q_n*,* q_p *and* q_{corr} *queries to oracles* NEW*,* PUNC *and* CORR*, respectively, there exists an adversary* $\mathcal{B}$ *running in approximately the same time as* $\mathcal{A}$ *and making at most* q_n *queries to its oracle* NEW-RO\$-WRAP*, such that*

$$\mathbf{Adv}^{\text{find\$-cpa}}_{\mathsf{PKW}}(\mathcal{A}) \leq q_{ro\$} \cdot \mathbf{Adv}^{\text{find\$-1cpa}}_{\mathsf{PKW}}(\mathcal{B}).$$

Proof. We show this implication via a hybrid argument over $\mathcal{A}$'s RO\$-WRAP queries per user key, yielding the factor $q_{ro\$}$. Each hybrid step is reduced to the find\$-1cpa security of PKW through a reduction $\mathcal{B}$, which relays a single RO\$-WRAP per key to the NEW-RO\$-WRAP in find\$-1cpa, and responds to all other queries with random resp. real wrappings. As the NEW-RO\$-WRAP does not allow the secret key to be punctured prior to the challenge wrap, $\mathcal{B}$ saves puncturing calls and performs them after receiving the challenge ciphertext and accompanying key. The soundness of this reordering relies on the puncture invariance and consistency of PKW. If $\mathcal{A}$ happens to puncture a key on a tag T and then ask RO\$-WRAP on the same tag T, $\mathcal{B}$ simulates the latter output with $\perp$, relying on the assumption that $\mathsf{Wrap}(sk, T, \cdot, \cdot)$ outputs $\perp$ after puncturing sk on T. (Note that this final assumption is what enables this reduction, in contrast to the general separation in Theorem 12.) □

Lastly, we show that access to real wraps in addition to the real-or-random challenge wraps results in a strictly stronger security notion for PKW schemes.

Theorem 14 (PKW: find\$-cpa $\not\implies$ find\$-rcpa)**.** *Assume there exists a* find\$-cpa *secure PKW scheme* PKW *with tag space* $\mathcal{T} = \{0,1\}^t$ *and a* fpr-ro\$ *secure PPRF* PF *with domain and range* $\mathcal{X} = \mathcal{Y} = \{0,1\}^t$*, for some positive integer* t*. Then there exists a PKW scheme* PKW' *which is* find\$-cpa *secure, but not* find\$-rcpa *secure.*

Proof. Let $\mathsf{PKW} = (\mathsf{KeyGen}, \mathsf{Wrap}, \mathsf{Unwrap}, \mathsf{Punc})$ be a PKW scheme with secret key space $\mathcal{SK}$, tag space $\mathcal{T} = \{0,1\}^t$, header space $\mathcal{H}$ wrap-key space $\mathcal{K} = \{0,1\}^k$ and ciphertext-length function $\mathsf{cl} : \mathbb{N} \to \mathbb{N}$. Let $\mathsf{PF} = (\mathsf{KeyGen}, \mathsf{Eval}, \mathsf{Punc})$ be a PPRF with key space $\mathcal{SK}_f$, domain and range $\mathcal{X} = \mathcal{Y} = \{0,1\}^t$. Let $\mathsf{PKW}' = (\mathsf{KeyGen}', \mathsf{Wrap}', \mathsf{Unwrap}', \mathsf{Punc}')$ be constructed from PKW as shown in Figure 15.

CONSTRUCTION IDEA. The idea of the separating example is to extend secret key sk' of PKW' to, beyond the secret key sk of PKW and a secret key sk_f for the PPRF PF, additionally include a copy of the original, unpunctured secret key sk_0, which can be used to distinguish a real challenge from a random one in the find\$-rcpa game. To ensure that the scheme is still find\$-cpa secure, the puncturing algorithm of PKW' removes the copy of the unpunctured secret key from sk' if $\mathsf{PKW}'.\mathsf{Punc}$ is called on any other input than a specific tag $\hat{T} := \mathsf{PF.Eval}(sk_f, 0^t)$. To enable the attack on the find\$-rcpa security of PKW', algorithm $\mathsf{PKW}'.\mathsf{Wrap}'(sk', T, H, K)$ is modified to prepend $\mathsf{PF.Eval}(sk_f, T)$ to the usual ciphertext produced by $\mathsf{PKW.Wrap}$, allowing $\hat{T}$ to be learned from a wrap on tag $T_0 = 0^t$.

ATTACK ON find\$-rcpa SECURITY. We begin by showing that PKW' is not find\$-rcpa secure. For this we present the strategy of an adversary $\mathcal{A}$ which achieves advantage

$$\mathbf{Adv}^{\text{find\$-rcpa}}_{\mathsf{PKW}'}(\mathcal{A}) \geq \left(1 - \mathbf{Adv}^{\text{fpr-ro\$}}_{\mathsf{PF}}(\mathcal{B}) - \frac{1}{2^t}\right)\left(1 - \frac{1}{2^{\mathsf{cl}(k)}}\right),$$

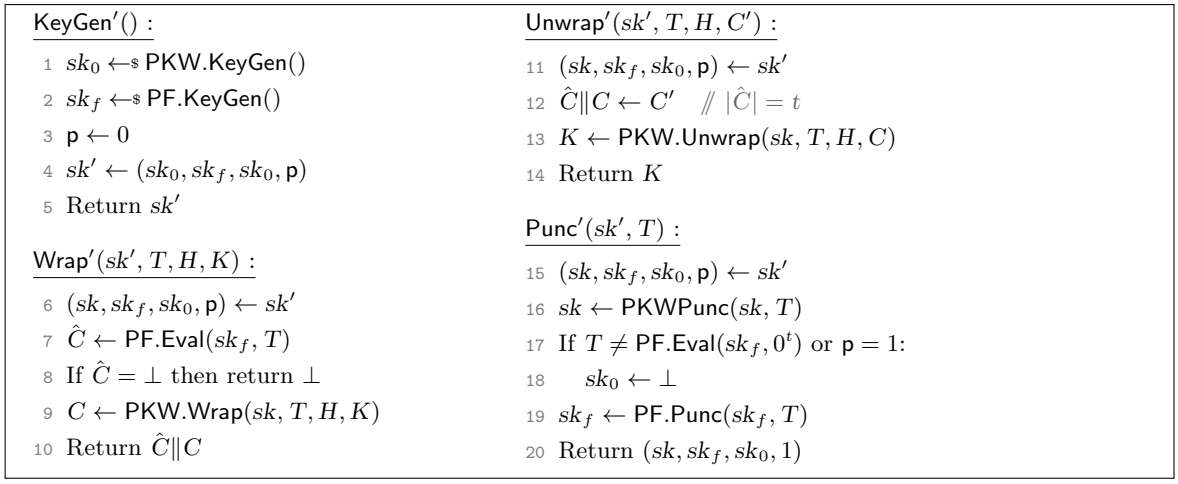

Figure 15: Construction of PKW scheme PKW' for Theorem 14. The construction is parameterized by $\mathsf{PKW} = (\mathsf{KeyGen}, \mathsf{Wrap}, \mathsf{Unwrap}, \mathsf{Punc})$ with tag space $\{0,1\}^t$ and $\mathsf{PF} = (\mathsf{KeyGen}, \mathsf{Eval}, \mathsf{Punc})$ with domain and range $\mathcal{X} = \mathcal{Y} = \{0,1\}^t$.

where adversary $\mathcal{B}$ is a simple distinguisher which checks if the probability that $\mathsf{PF.Eval}(sk_f, 0^t) = 0^t$ deviates from $\frac{1}{2^t}$. (E.g. by running "If RO$-EVAL$(0^t) = 0^t$ then return 1, else return 0".)

Adversary $\mathcal{A}$ works as follows. First, it queries oracle NEW() to initialize key sk'_1. It then issues query $\hat{T} \| C_1 \leftarrow$ WRAP$(1, 0^t, H_1, K_1)$ for an arbitrary header H_1 and wrap key K_1. Next, it queries $\hat{C} \| C_2 \leftarrow$ RO$-WRAP$(1, \hat{T}, H_2, K_2)$ for another (not necessarily different) arbitrary header H_2 and key K_2 and then punctures sk'_1 on $\hat{T}$ by calling PUNC$(1, \hat{T})$. After this, it may corrupt key sk'_1 using oracle CORR(1). By definition of PKW', this strategy ensures that the copy sk_0 of the original, unpunctured secret key is still part of sk'_1. $\mathcal{A}$ may then use sk_0 to reproduce the challenge ciphertext by locally running $\mathsf{PKW.Wrap}(sk_0, \hat{T}, H_2, K_2)$ and comparing the result to C_2. If they are the same, $\mathcal{A}$ halts and returns 1, otherwise 0.

Before analyzing the advantage of $\mathcal{A}$, we note that some care has to be taken if $\hat{T}$ happens to be equal to 0^t, since the adversary is required to be tag-respecting, meaning that the game will replace the challenge response with $\perp$ if the tag is repeated. Hence the attack fails in the low-probability event bad that $\hat{T}_1 = 0^t$. This occurs with probability $\Pr[\,\mathsf{bad}\,] \leq \mathbf{Adv}^{\text{fpr-ro\$}}_{\mathsf{PF}}(\mathcal{B}) + \frac{1}{2^t}$, where $\mathcal{B}$ is defined as above.

Let b denote the secret bit drawn by the challenger in the game. The strategy described gives $\Pr\left[\,\mathcal{A} \Rightarrow 1 \,\middle|\, b = 1, \neg\mathsf{bad}\,\right] = 1$ and $\Pr\left[\,\mathcal{A} \Rightarrow 1 \,\middle|\, b = 0, \neg\mathsf{bad}\,\right] = \frac{1}{2^{\mathsf{cl}(k)}}$, since the challenge ciphertext C_2 is drawn uniformly at random from $\{0,1\}^{\mathsf{cl}(k)}$ if $b = 0$. This gives the claimed lower bound on the advantage of $\mathcal{A}$.

PROVING THE find$-cpa SECURITY OF PKW'. Next, we show that PKW' is find$-cpa secure, given that PKW is find$-cpa secure and PF is fpr-ro$ secure. For every adversary $\mathcal{A}$ making at most q_n and $q_{ro\$}$ queries to oracle NEW and RO$-WRAP, respectively, we design adversaries $\mathcal{B}_{\mathsf{pkw}}$ and $\mathcal{B}_{\mathsf{pprf}}$, running in approximately the same time as $\mathcal{A}$ and making at most $q_{ro\$}$ queries to oracle RO$-WRAP and RO$-EVAL, respectively, such that

$$\mathbf{Adv}^{\text{find\$-cpa}}_{\mathsf{PKW}'}(\mathcal{A}) \leq 2 \cdot \mathbf{Adv}^{\text{fpr-ro\$}}_{\mathsf{PF}}(\mathcal{B}_{\mathsf{pprf}}) + 2 \cdot \frac{2q_n}{2^t} + \mathbf{Adv}^{\text{find\$-cpa}}_{\mathsf{PKW}}(\mathcal{B}_{\mathsf{pkw}}). \tag{4}$$

The proof proceeds by a series of game hops. In the first hop from game $\mathbf{G}_0 := \mathbf{G}^{\text{find\$-cpa}}_{\mathsf{PKW}'}$ to game $\mathbf{G}_1$, the result of all PPRF evaluations are replaced by consistent random strings. We design $\mathcal{B}_{\mathsf{pprf}}$ such that $|\Pr[\,\mathbf{G}_0\,] - \Pr[\,\mathbf{G}_1\,]| \leq \mathbf{Adv}^{\text{fpr-ro\$}}_{\mathsf{PF}}(\mathcal{B}_{\mathsf{pprf}})$, as follows. Adversary $\mathcal{B}_{\mathsf{pprf}}$ simulates game $\mathbf{G}_0$ for $\mathcal{A}$ by drawing a bit b' and using the oracles in the fpr-ro$ game to act as the challenger in game $\mathbf{G}_0$ would with secret bit b', except that PPRF evaluations under key i on tag T are replaced by the response to a query RO$-EVAL$(i, T)$. When $\mathcal{A}$ halts and outputs $b^*_{\mathcal{A}}$, $\mathcal{B}_{\mathsf{pprf}}$ halts and returns 1 if $b' = b^*_{\mathcal{A}}$, else 0. Let b denote the random bit drawn by the challenger in the fpr-ro$ game, then adversary $\mathcal{B}_{\mathsf{pprf}}$ simulates game $\mathbf{G}_0$ for $\mathcal{A}$ when $b = 1$ and game $\mathbf{G}_1$ when $b = 0$. Because the same restriction on CORR queries exists in both games, the simulation is sound.

In the second hop, we exclude the possibility that $\mathcal{A}$ corrupts a key which still includes the copy of the corresponding unpunctured key. That is, for each key index $i \in \{1, \ldots, q_n\}$, we let bad_i be the event that either (1) $\mathcal{A}$ *guessed* the (by now uniformly random) special tag $\hat{T}_i$ in its first query to oracle

RO$-WRAP or (2) that $\hat{T}_i = 0^t$ and adversary $\mathcal{A}$ learns this in its first RO$-WRAP query. The probability of each of these events is $\frac{1}{2^t}$, since $\hat{T}_i$ is drawn u.a.r. in $\{0,1\}^t$ for each i, giving $\Pr[\mathsf{bad}_i] \leq \frac{2}{2^t}$. Let $\mathsf{bad} = \mathsf{bad}_1 \vee \ldots \vee \mathsf{bad}_{q_n}$, then by the union bound $\Pr[\mathsf{bad}] \leq \frac{2q_n}{2^t}$. Let games $\mathbf{G}_1$ and $\mathbf{G}_2$ be identical-until-bad, such that in game $\mathbf{G}_2$ the unpunctured copy of the secret key is overwritten by $\perp$ before a corruption query is responded to if bad has occured. This ensures that in game $\mathbf{G}_2$, unpunctured copy of the secret key is never given out to adversary $\mathcal{A}$. By the fundamental lemma of game-playing [7], this gives $|\Pr[\mathbf{G}_1] - \Pr[\mathbf{G}_2]| \leq \Pr[\mathsf{bad}] \leq \frac{2q_n}{2^t}$.

Lastly, we construct an adversary $\mathcal{B}_{\mathsf{pkw}}$ such that $2 \cdot \left|\Pr[\mathbf{G}_2(\mathcal{A})] - \frac{1}{2}\right| \leq \mathbf{Adv}^{\text{find\$-cpa}}_{\mathsf{PKW}}(\mathcal{B}_{\mathsf{pkw}})$. The reduction is straightforward: adversary $\mathcal{B}_{\mathsf{pkw}}$ relays any queries from $\mathcal{A}$ to its own corresponding oracles, and supplements the responses with consistent random strings (e.g. prepended to challenge ciphertexts) where needed. To simulate a corruption query on key index i, $\mathcal{B}_{\mathsf{pkw}}$ queries oracle CORR(i) to obtain the current PKW key sk_i. It also generates a PPRF key $sk_{f,i}$ and punctures it on any tags which $\mathcal{A}$ has queried oracle PUNC on. It then returns $(sk_i, sk_{f,i}, \perp, \mathsf{p})$ to adversary $\mathcal{A}$. The previous game hop ensures that this is a sound simulation, as the corruption oracle in game $\mathbf{G}_2$ always returns a key where the third component is $\perp$. When $\mathcal{A}$ halts and returns b^*, $\mathcal{B}_{\mathsf{pkw}}$ also halts and returns b^*. This gives the claimed advantage bound.

Putting everything together yields Equation (4). □

C All-in-One Notions for PKW

In [51], Rogaway and Shrimpton give a combined confidentiality and integrity notion for key-wrapping schemes, which they call "DAE security" (deterministic authenticated encryption). They show that such a combined notion is equivalent to two separate notions in their setting, as well as for authenticated encryption in general. Here, we translate this result to the PKW setting and formally confirm that a similar equivalence holds also for our notions capturing forward security.

In Figure 16, we provide a combined confidentiality and integrity notion for PKW schemes. The games $\mathbf{G}^{\text{find\$int}}$ and $\mathbf{G}^{\text{find\$int-r}}$ capture the amalgamation of find$-cpa security (with real wrappings) and integrity of ciphertexts. The notions can be viewed as extensions of find$-cpa and find$-rcpa, where the adversary additionally gets access to an unwrapping oracle, which in the real world returns honestly unwrapped keys (from tags, headers, and ciphertexts chosen by the adversary) and in the ideal world always returns $\perp$. We name the new oracle "real-or-bot unwrap": RO$\perp$-UNWRAP. To exclude trivial attacks, queries to RO$\perp$-UNWRAP are only permitted under keys which have not (yet) been compromised through a query to oracle CORR.

The advantage of an adversary against the find$int (find$int-r) security of a PKW scheme is defined as usual.

Definition 15 (PKW confidentiality + integrity (find$int, find$int-r))**.** Let PKW be a puncturable key-wrapping scheme. We define the advantage of an adversary $\mathcal{A}$ against the forward indistinguishability and integrity $X \in \{\text{find\$int}, \text{find\$int-r}\}$ of PKW as

$$\mathbf{Adv}^{X}_{\mathsf{PKW}}(\mathcal{A}) = 2\left|\Pr\left[\mathbf{G}^{X}_{\mathsf{PKW}}(\mathcal{A}) \Rightarrow \text{true}\right] - \frac{1}{2}\right|,$$

where $\mathbf{G}^{X}_{\mathsf{PKW}}(\mathcal{A})$ is defined in Figure 16.

Next, in Theorems 15 and 16, we show that the combination of confidentiality (Definition 10) and integrity of ciphertexts (Definition 11) of PKW schemes as separate properties is equivalent to find$int-r security, as expected. In the forward direction (find$-rcpa + int-ctxt $\implies$ find$int-r, shown in Theorem 15), the reduction to PKW int-ctxt security induces a loss proportional to the number of new key queries made by the adversary in the find$int-r game. The loss stems from the find$int-r game including a corruption oracle CORR to capture forward-secure confidentiality, whereas the integrity game does not have such a corruption oracle. This is not a problem in the single-key setting, since find$int-r does not permit unwrapping queries after key compromise. Hence the integrity adversary in the reduction can abort the game if it receives a query to oracle CORR, since successful forgeries are excluded from occurring after a corruption query. However, in the multi-key setting, a CORR query does not prevent a later forgery under a different key. Since the integrity adversary does not know in advance under which key the first successful forgery will occur, it needs to treat all keys as challenge keys and relay wraps and unwraps to the oracles in the integrity game. This creates a key-commitment problem which prevents the reduction from simply sampling a fresh key when forced to respond to a key-corruption

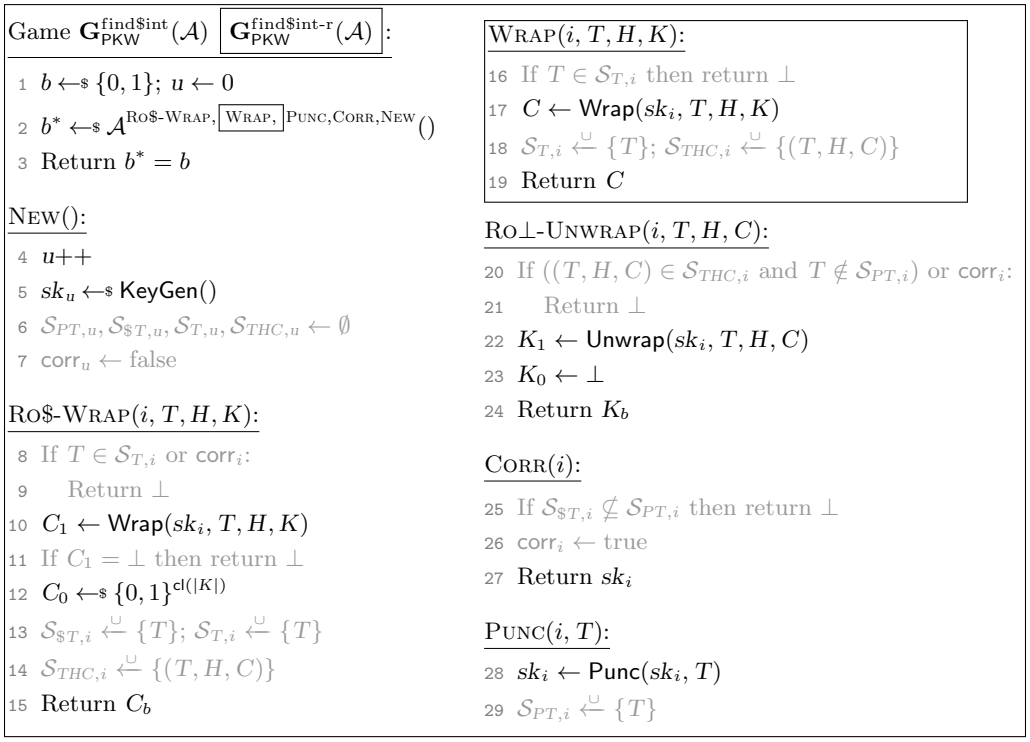

Figure 16: Forward indistinguishability and integrity find$int / find$int-r of a puncturable key-wrapping scheme PKW. The code in boxes is executed in $\mathbf{G}^{\text{find\$int-r}}_{\mathsf{PKW}}$, but not in $\mathbf{G}^{\text{find\$int}}_{\mathsf{PKW}}$.
Grey code prevents trivial attacks and ensures that the queries are tag-respecting.

query. To circumvent this, the reduction to the int-ctxt notion guesses the key index under which the first successful forgery is made, resulting in the aforementioned loss.

Theorem 15 says that scheme PKW is find$int-r (or find$int) secure, given that it is find$-rcpa (or find$-cpa) and int-ctxt secure.

Theorem 15 (PKW: find$-rcpa/find$-cpa + int-ctxt $\implies$ find$int-r/find$int)**.** *Let* $\mathsf{PKW} = (\mathsf{KeyGen}, \mathsf{Wrap}, \mathsf{Unwrap}, \mathsf{Punc})$ *be a PKW scheme. Then for every adversary* $\mathcal{A}$ *(and every* $\mathcal{A}'$*) against the* find$int-r *(or* find$int*) security of* PKW *making at most* q_n *to oracle* New*, there exist adversaries* $\mathcal{B}$ *and* $\mathcal{C}$ *(*$\mathcal{B}'$ *and* $\mathcal{C}'$*) running in approximately the same time and using the same resources*[10] *as* $\mathcal{A}$ *(or* $\mathcal{A}'$*, respectively) , such that*

$$\mathbf{Adv}^{\text{find\$int-r}}_{\mathsf{PKW}}(\mathcal{A}) \le 2q_n \cdot \mathbf{Adv}^{\text{int-ctxt}}_{\mathsf{PKW}}(\mathcal{B}) + \mathbf{Adv}^{\text{find\$-rcpa}}_{\mathsf{PKW}}(\mathcal{C}) \qquad \textit{and}$$
$$\mathbf{Adv}^{\text{find\$int}}_{\mathsf{PKW}}(\mathcal{A}') \le 2q_n \cdot \mathbf{Adv}^{\text{int-ctxt}}_{\mathsf{PKW}}(\mathcal{B}') + \mathbf{Adv}^{\text{find\$-cpa}}_{\mathsf{PKW}}(\mathcal{C}').$$

Proof. The result for find$int security holds analogously by simply ignoring any mention of oracle Wrap in the final step of the proof; hence we focus on find$int-r. The proof has one game hop, starting from $\mathbf{G}_0 = \mathbf{G}^{\text{find\$int-r}}_{\mathsf{PKW}}$. Let $\mathbf{Adv}_i(\mathcal{A}) := 2\left|\Pr[\,\mathbf{G}_i(\mathcal{A}) \Rightarrow \text{true}\,] - \frac{1}{2}\right|$ for $i \in \{0,1\}$.

$\mathbf{G}_0 \to \mathbf{G}_1$: In $\mathbf{G}_1$, we set a flag bad $\leftarrow$ true if $K_1 \neq \perp$ in any Ro$\perp$-Unwrap query, and if bad is set overwrite $K_1 \leftarrow \perp$. All Ro$\perp$-Unwrap queries are hence replied to with $\perp$, independent of the secret bit b. Since the games are identical-until-bad, $|\Pr[\,\mathbf{G}_0(\mathcal{A})\,] - \Pr[\,\mathbf{G}_1(\mathcal{A})\,] \le \Pr[\,\mathsf{bad}\,]$ for all adversaries $\mathcal{A}$.

We bound $\Pr[\,\mathsf{bad}\,] \le q_n \cdot \mathbf{Adv}^{\text{int-ctxt}}_{\mathsf{PKW}}(\mathcal{B})$ by an adversary $\mathcal{B}$ which simulates $\mathbf{G}_0$ for $\mathcal{A}$ as the challenger (sampling a challenge bit b'). Initially, $\mathcal{B}$ guesses the key index i under which the *first* query to Ro$\perp$-Unwrap is made which sets bad. If the guess for i turns out to be wrong, $\mathcal{B}$ aborts, introducing a loss in the number of keys q_n. For all key indices $j \neq i$, $\mathcal{B}$ generates sk_j and answers all oracle queries itself. For key index i, it relays Wrap and Punc queries to the corresponding oracles in the int-ctxt game. If $b' = 1$, it forwards Ro$-Wrap queries to oracle Wrap. Otherwise $\mathcal{B}$ samples a random ciphertext of the correct length and returns it to $\mathcal{A}$. In simulating oracle Ro$\perp$-Unwrap, adversary $\mathcal{B}$ forwards the query to its Unwrap oracle. If $\mathcal{A}$ tries to corrupt key sk_i by querying oracle Corr(i), then $\mathcal{B}$ aborts the simulation.

[10] The exact number of queries made by each is evident from the proof.

Adversary $\mathcal{B}$ perfectly simulates game $\mathbf{G}_0$ for $\mathcal{A}$ when targeting key index i with its first forgery, unless $\mathcal{A}$ tries to compromise key sk_i. However bad can only occur for key index i if sk_i has not been compromised (i.e., if corr_i = false). If event bad occurs and $\mathcal{B}$ guessed key index i correctly, then adversary $\mathcal{A}$ has successfully forged a valid ciphertext and submitted it to oracle RO⊥-UNWRAP, and since $\mathcal{B}$ forwards all such queries to its challenger, $\mathcal{B}$ has consequently set the win flag to true in game $\mathbf{G}^{\text{int-ctxt}}_{\mathsf{PKW}}$. Hence $\Pr[\,\mathsf{bad}\,] \leq q_n \cdot \mathbf{Adv}^{\text{int-ctxt}}_{\mathsf{PKW}}(\mathcal{B})$. This gives $\mathbf{Adv}_0 \leq 2q_n \cdot \mathbf{Adv}^{\text{int-ctxt}}_{\mathsf{PKW}}(\mathcal{B}) + \mathbf{Adv}_1$.

$\mathbf{G}_1$: The advantage of any adversary $\mathcal{A}$ in game $\mathbf{G}_1$ can now be bounded by the advantage $\mathbf{Adv}^{\text{find\$-rcpa}}_{\mathsf{PKW}}(\mathcal{C})$ of an adversary $\mathcal{C}$ against the find\$-rcpa security of PKW. The reduction is straightforward: $\mathcal{C}$ simply relays all queries to oracles RO\$-WRAP, WRAP, PUNC and CORR to its own corresponding oracles. It responds to all RO⊥-UNWRAP queries with ⊥. When $\mathcal{A}$ halts and outputs a bit b^*, $\mathcal{C}$ halts and returns b^* to its challenger. This way $\mathcal{C}$ perfectly simulates game $\mathbf{G}_1$ for $\mathcal{A}$ and $\mathbf{Adv}_1(\mathcal{A}) \leq \mathbf{Adv}^{\text{find\$-rcpa}}_{\mathsf{PKW}}(\mathcal{C})$. □

Next, we complete the equivalence by showing that find\$int-r (or find\$int) security implies find\$-rcpa (or find\$int) and int-ctxt security.

Theorem 16 (PKW: find\$int-r/find\$int $\implies$ find\$-rcpa/find\$-cpa + int-ctxt)**.** *Let* $\mathsf{PKW} = (\mathsf{KeyGen}, \mathsf{Wrap}, \mathsf{Unwrap}, \mathsf{Punc})$ *be a PKW scheme. Then for any adversaries* $\mathcal{A}_1$*,* $\mathcal{A}_2$*,* $\mathcal{A}_3$ *against the* find\$-rcpa*,* find\$-cpa*, or* int-ctxt *security of* PKW*, there exist adversaries* $\mathcal{B}_1$*,* $\mathcal{B}_2$*,* $\mathcal{B}_3$ *and* $\mathcal{B}'_3$ *running in approximately the same time and using the same resources as* $\mathcal{A}_1$*,* $\mathcal{A}_2$*, and* $\mathcal{A}_3$*, respectively, such that*

$$\begin{aligned}
\mathbf{Adv}^{\text{find\$-rcpa}}_{\mathsf{PKW}}(\mathcal{A}_1) &\leq \mathbf{Adv}^{\text{find\$int-r}}_{\mathsf{PKW}}(\mathcal{B}_1),\\
\mathbf{Adv}^{\text{find\$-cpa}}_{\mathsf{PKW}}(\mathcal{A}_2) &\leq \mathbf{Adv}^{\text{find\$int}}_{\mathsf{PKW}}(\mathcal{B}_2), \qquad \textit{resp.}\\
\mathbf{Adv}^{\text{int-ctxt}}_{\mathsf{PKW}}(\mathcal{A}_3) &\leq \mathbf{Adv}^{\text{find\$int}}_{\mathsf{PKW}}(\mathcal{B}_3) \leq \mathbf{Adv}^{\text{find\$int-r}}_{\mathsf{PKW}}(\mathcal{B}'_3).
\end{aligned}$$

Proof. The first two results are trivial, as the reductions $\mathcal{B}_1$ and $\mathcal{B}_2$ against the find\$int-r/find\$int security of scheme PKW can simply ignore the RO⊥-UNWRAP to simulate the find\$-rcpa/find\$-cpa game.

For integrity, we do a single game hop, from $\mathbf{G}_0 = \mathbf{G}^{\text{int-ctxt}}_{\mathsf{PKW}}$ to a game $\mathbf{G}_1$ where the WRAP returns randomly sampled ciphertexts and the UNWRAP never sets the win flag and always returns ⊥ on all queries. By definition, $\mathbf{Adv}^{\text{int-ctxt}}_{\mathsf{PKW}} = \Pr[\,\mathbf{G}_0(\mathcal{A}_3)\,]$. Further, $\Pr[\,\mathbf{G}_1(\mathcal{A}_3)\,] = 0$, since $\mathcal{A}_3$ cannot trigger a win anymore.

It remains to bound $|\Pr[\,\mathbf{G}_0(\mathcal{A}_3)\,] - \Pr[\,\mathbf{G}_1(\mathcal{A}_3)\,]|$, which we do by a straightforward reduction $\mathcal{B}_3$ to find\$int security. Adversary $\mathcal{B}_3$ simulates oracles NEW and PUNC by relaying queries to the corresponding oracle in the find\$int game, while relaying WRAP queries to its RO\$-WRAP oracle and UNWRAP queries to its RO⊥-UNWRAP oracle. This way, $\mathcal{B}_3$ simulates game $\mathbf{G}_0$ for $\mathcal{A}_3$ when the hidden bit b in game $\mathbf{G}^{\text{find\$int}}_{\mathsf{PKW}}$ is 1, and $\mathbf{G}_1$ when $b = 0$. Thus $|\Pr[\,\mathbf{G}_0(\mathcal{A}_3)\,] - \Pr[\,\mathbf{G}_1(\mathcal{A}_3)\,]\,| \leq \mathbf{Adv}^{\text{find\$int}}_{\mathsf{PKW}}(\mathcal{B}_3)$, as required. Trivially also $\mathbf{Adv}^{\text{find\$int}}_{\mathsf{PKW}}(\mathcal{B}_3) \leq \mathbf{Adv}^{\text{find\$int-r}}_{\mathsf{PKW}}(\mathcal{B}'_3)$, by simply ignoring the WRAP oracle. □

D TLS Session Resumption: A Formal Violation of Integrity

We demonstrate how an omitted property in the definition of a PPRF in [2] leads to an issue with the integrity of their session resumption protocol. We stress that this issue can be easily fixed through an added assumption (or, as we propose, via stronger correctness requirements of a PPRF). We therefore view the resulting, theoretical attack in their formal model mainly as showcasing the need for careful definitions, and showing how suitable abstractions (such as PKW schemes in this case) can help simplify security proofs to avoid issues like this one.

The formal attack. Figure 17 shows an attack on the formal 0-RTT-SR security [2, Section 2.1] of the session resumption protocol $\mathsf{Resumption}$ as defined in [2, Section 3.2]. The attack is possible because the PPRF definition presented in AGJ is too weak and allows the PPRF to output a predictable point in the range instead of ⊥ after puncturing. It additionally highlights a bug in the proof of Theorem 3 in [2]: In the hop from game 2 to game 3, the PPRF evaluation on a specific point ν is replaced by a fixed value, regardless of whether the point is then later punctured on. This can easily be fixed by adapting the output to be ⊥, *if* a stronger PPRF definition is used, which explicitly demands that the output of Eval is ⊥ after puncturing, as we do.

Adversary $\mathcal{A}^{\textsc{Dec},\textsc{Test},\textsc{Corr}}(t_1, t_2, \ldots, t_\mu)$:

// $t_i = (i, C_i)$ where i is an integer and $C_i = \mathsf{AEAD.Enc}(\mathsf{F}'.\mathsf{Eval}(sk_p, i), K_i, i)$

$K^* \leftarrow 0^k$ // Arbitrary session key

$C^* \leftarrow\!\!\$\ \mathsf{AEAD.Enc}(0^{out\ell}, K^*, 1)$

$t^* \leftarrow (1, C^*)$

$K_1 \leftarrow \textsc{Dec}(t_1)$ // Now sk_p is punctured on 1

$K \leftarrow \textsc{Test}(t^*)$

// If $b = 1$ then $K = \mathsf{ServerRes}((sk_p, \mu), (1, C^*))$
$= \mathsf{AEAD.Dec}(\mathsf{F}'.\mathsf{Eval}(sk_p, 1), C^*, 1)$
$= \mathsf{AEAD.Dec}(0^{out\ell}, C^*, 1) = K^*$ by AEAD correctness

// If $b = 0$ then K is a random element in $\mathcal{S}$

If $K = K^*$ then return 1

Return 0

Figure 17: Attack showcasing the integrity issue in the generic construction of a session resumption protocol by [2].

The attack makes use of a specifically crafted PPRF which fulfills the "rand" security requirement of AGJ [2, Definition 7]. It is constructed as follows. Let $\mathsf{F} = (\mathsf{KeyGen}, \mathsf{Eval}, \mathsf{Punc})$ be a rand secure PPRF with domain $\mathcal{X} = \{0,1\}^{in\ell}$ and range $\mathcal{Y} = \{0,1\}^{out\ell}$, that *fulfills our correctness requirements on PPRFs* (Def. 4, point (4)).[11] Let $\mathsf{F}' = (\mathsf{KeyGen}, \mathsf{Eval}', \mathsf{Punc})$ be identical to F, except that the evaluation algorithm Eval' is modified to return $0^{out\ell}$ whenever Eval would return $\perp$. That is, for a secret key sk and a point $x \in \mathcal{X}$, $\mathsf{Eval}'(sk, x) := 0^{out\ell}$ if $\mathsf{Eval}(sk, x) = \perp$, otherwise $\mathsf{Eval}'(sk, x) := \mathsf{Eval}(sk, x)$.

We show that F' is rand secure by a reduction to the fpr-rro$ security of F. That is, for all adversaries $\mathcal{A}$ there exists an adversary $\mathcal{B}$ such that

$$\mathbf{Adv}^{\text{rand}}_{\mathsf{F}'}(\mathcal{A}) \leq \mathbf{Adv}^{\text{fpr-rro\$}}_{\mathsf{F}}(\mathcal{B}).$$

Adversary $\mathcal{B}$ simulates game rand for adversary $\mathcal{A}$, using the oracles in the fpr-rro$ game to act as the challenger. Since the rand game uses the original, unpunctured secret key for all evaluations, the difference between F and F' will not be visible to the adversary during the game, so no adaptations are necessary to account for this. Adversary $\mathcal{B}$ uses the final output bit b^* of $\mathcal{A}$ as its own guess of the secret bit in game fpr-rro$.

Conditioned on the challenge point returned by $\mathcal{A}$ in the first stage being a valid challenge (i.e. not used in a prior query to the evaluation oracle), adversary $\mathcal{B}$ perfectly simulates the rand game for $\mathcal{A}$ with secret bit b, where b is the secret bit in the fpr-rro$ game. Let $\mathcal{Q}$ denote the set of points queried to oracle $\textsc{Eval}$ by $\mathcal{A}$. Then

$$\begin{aligned}\Pr\left[\,\mathbf{G}^{\text{rand}}_{\mathsf{F}'} \Rightarrow \text{true}\,\right] &= \Pr\left[\,b^* = b_{\text{rand}} \wedge x^* \notin \mathcal{Q}\,\right] \\ &\leq \Pr\left[\,b^* = b_{\text{rand}} \,\middle|\, x^* \notin \mathcal{Q}\,\right] \leq \Pr\left[\,b^* = b\,\right] = \Pr\left[\,\mathbf{G}^{\text{fpr-rro\$}}_{\mathsf{F}}(\mathcal{B}) \Rightarrow \text{true}\,\right].\end{aligned}$$

Next we explain the attack on the 0-RTT-SR security of session resumption protocol $\mathsf{Resumption}[\mathsf{F}', \mathsf{AEAD}] = (\mathsf{Setup}, \mathsf{TicketGen}, \mathsf{ServerRes})$ built from F' and a *randomized* AEAD scheme AEAD, as per the construction by AGJ [2, Section 3.2]. Let $\mathcal{S} = \{0,1\}^k$ be the session key space of $\mathsf{Resumption}$. Let $sk = (sk_p, n)$ denote the secret key drawn during setup. The attack, spelled out in Figure 17, makes use of oracles $\textsc{Dec}(t)$ (running $\mathsf{ServerRes}$ to decrypt ticket t and return the resulting session key) and $\textsc{Test}(t)$ (running $\mathsf{ServerRes}$ to decrypt ticket t and return the resulting session key if the secret bit in the game is 1, else a random session key from $\mathcal{S}$). If $b = 1$ then the attack succeeds with probability 1. If $b = 0$ then $\Pr[\,K = K^*\,] = \frac{1}{2^k}$ since K is drawn uniformly at random in $\mathcal{S}$ by the challenger. Hence

$$\mathbf{Adv}^{\text{0-RTT-SR}}_{\mathsf{Resumption},\mu}(\mathcal{A}) = 2\left|\Pr\left[\,\mathbf{G}^{\text{0-RTT-SR}}_{\mathsf{Resumption},\mu}(\mathcal{A}) \Rightarrow 1\,\right] - \frac{1}{2}\right| = 1 - \frac{1}{2^k}.$$

E PFS Instantiation Proofs

[11] That is, in particular $\mathsf{F.Eval}(sk, x)$ always returns $\perp$ if sk has been punctured on x.

E.1 Proof of Theorem 7: PFS[PKW, AEAD] is find$-rcpa, via PKW find$-1cpa

The proof proceeds by a series of game hops, starting with game $\mathbf{G}_0$, which is equivalent to $\mathbf{G}^{\text{find\$-rcpa}}_{\mathsf{PFS[PKW,AEAD]}}$, and ending in game $\mathbf{G}_6$, where the distribution of the output components of the challenge oracle RO$-ENC is independent of the challenge bit b. For any adversary $\mathcal{A}$, we define $\mathbf{Adv}_i(\mathcal{A}) := 2\left|\Pr[\,\mathbf{G}_i(\mathcal{A})\,] - \frac{1}{2}\right|$ for $i \in \{0,\ldots,6\}$. Hence $\mathbf{Adv}^{\text{find\$-rcpa}}_{\mathsf{PFS[PKW,AEAD]}}(\mathcal{A}) = \mathbf{Adv}_0(\mathcal{A})$.

$\mathbf{G}_0 \to \mathbf{G}_1$: In the first game hop, file identifiers for challenge encryptions are prevented from being drawn from the set of already shredded identifiers, both in the ideal world (within the game) and in the real world (within the PFS[PKW, AEAD] construction). Without this exclusion, the output from oracle RO$-ENC in the real world is undefined if the file id has been previously shredded (since the output of PKW wrapping using a previously punctured-on tag is undefined), whereas in the ideal world nothing prevents a file id from appearing again after being shredded.

Let bad_1 be the event that in any of the at most $q_{ro\$}$ many queries to oracle RO$-ENC, the ideal or the real world file identifier (id_0, id_1) is drawn to be one of the at most q_s many shredded file identifiers under the current secret key. Let game $\mathbf{G}_1$ be equivalent to game $\mathbf{G}_0$ unless bad_1 occurs. If bad_1 occurs, the RO$-ENC query, and all subsequent queries, are responded to by $\perp$. Then $|\Pr[\,\mathbf{G}_0\,] - \Pr[\,\mathbf{G}_1\,]| \leq \Pr[\,\mathsf{bad}_1\,] \leq 2q_{ro\$} \cdot \frac{q_s}{2^t}$, since two identifiers are drawn u.a.r. from $\{0,1\}^t$ in each of the $q_{ro\$}$ queries, each of which has a probability of hitting one of the shredded identifiers of at most $q_s/2^t$. Hence, for all adversaries $\mathcal{A}$:

$$\mathbf{Adv}_0(\mathcal{A}) \leq 4q_{ro\$} \cdot \frac{q_s}{2^t} + \mathbf{Adv}_1(\mathcal{A}), \tag{5}$$

where the additional factor 2 stems from the scaling in the advantage definition.

$\mathbf{G}_1 \to \mathbf{G}_2$: In the second game hop, the adversary is limited to making a single challenge encryption query. That is, game $\mathbf{G}_2$ is identical to game $\mathbf{G}_1$, except that the adversary is restricted to making at most one query to the challenge oracle RO$-ENC. By applying a hybrid argument where the single challenge query in game $\mathbf{G}_2$ is used to move step-wise from game $\mathbf{H}_0 := \mathbf{G}_1$ with all challenge queries responded to with random bits (i.e. secret bit $b = 0$) to game $\mathbf{H}_{q_{ro\$}} := \mathbf{G}_1$ with all challenges responded to with real encryptions (secret bit $b = 1$), we obtain for all adversaries $\mathcal{A}$

$$\mathbf{Adv}_1(\mathcal{A}) = q_{ro\$} \cdot \mathbf{Adv}_2(\mathcal{A}') \tag{6}$$

for an adversary $\mathcal{A}'$ making at most one RO$-WRAP query, at most $q_e + q_{ro\$} - 1$ queries to ENC and at most $m - 1$ queries to oracle ROTKEY in game $\mathbf{G}_2$. The reduction works as follows.

Adversary $\mathcal{A}'$ draws an index $i \in \{1, 2, \ldots, q_{ro\$}\}$ u.a.r. and simulates hybrid game $\mathbf{H}_{i-1}$ or $\mathbf{H}_i$ for $\mathcal{A}$ using the oracles in $\mathbf{G}_2$. ENC, SHRED and CORR queries are relayed by $\mathcal{A}'$ to its own corresponding oracles. For queries to oracle RO$-ENC, the first $i-1$ are responded to using the real encryption oracle of $\mathcal{A}'$, the ith query is relayed to the RO$-ENC of $\mathcal{A}'$ and the remaining are responded to with random strings from the appropriate spaces. This way $\mathcal{A}'$ simulates $\mathbf{H}_{i-1}$ if it is in the ideal world in $\mathbf{G}_2$ and $\mathbf{H}_i$ if it is in the real world. In order for the simulation to be perfect, adversary $\mathcal{A}'$ queries oracle ENC also when generating the random responses to challenge queries by $\mathcal{A}$, to check that the result is not $\perp$. It then replaces the result by random before returning it to adversary $\mathcal{A}$. For this reason, and because adversary $\mathcal{A}'$ makes only a single query to its RO$-ENC challenge oracle (in contrast to up to $q_{ro\$}$ many by $\mathcal{A}$), the lists R and Q kept by the game will be different in the simulation than in the original game. Hence adversary $\mathcal{A}'$ needs to do its own bookkeeping and modify the output of oracle ROTKEY before returning it to $\mathcal{A}$. Specifically, headers in Q generated to simulate real challenge encryptions need to be moved to the beginning of R, and entries in Q generated for random challenges to check that the result is not $\perp$ need to be removed altogether. The file identifiers and new headers of the random challenges generated by $\mathcal{A}'$ need to be added to the end of R. This is easy thanks to the file identifiers which are unchanged by the key rotation. Note also that in order for the simulation to be perfect despite the input to the key rotation algorithm being different than in the original game, we rely on the fact that in our construction the new secret key is independent of the input to algorithm RotKey.

In game $\mathbf{G}_2$, we may assume w.l.o.g. that adversary $\mathcal{A}'$ makes *exactly* one query to oracle RO$-WRAP, as without any challenge query the secret bit is completely independent from the game.

$\mathbf{G}_2 \to \mathbf{G}_3$: In this game hop, the file identifiers (tags for the PKW scheme) generated by real encryption queries prior to the challenge query are excluded from coinciding with the file id of the challenge encryption. Let bad_2 be the event that such a collision occurs, and let $\mathbf{G}_3$ function like $\mathbf{G}_2$, except that the file

id of the challenge encryption is drawn already at the start of the game and saved until the adversary makes the query to RO$-ENC. Since it is drawn uniformly at random, this makes no difference for the distribution, but it allows the game to check whether the bad event occurs. If bad_2 happens in game $\mathbf{G}_3$, the current and all subsequent queries are responded to by $\perp$.

Since games $\mathbf{G}_2$ and $\mathbf{G}_3$ are identical-until-bad, we have

$$|\Pr[\mathbf{G}_2] - \Pr[\mathbf{G}_3]| \leq \Pr[\mathsf{bad}_2] \leq \frac{q_e + q_{ro\$} - 1}{2^t},$$

where the bound on the probability of bad_2 is due to there being at most $q_e + q_{ro\$} - 1$ real encryption queries after the hybrid, in each of which the file id is drawn independently and u.a.r. from $\{0,1\}^t$, making the probability of hitting the file id of the challenge 2^{-t} for each. Thus, for any adversary $\mathcal{A}$,

$$\mathbf{Adv}_2(\mathcal{A}) = 2 \cdot \frac{q_e + q_{ro\$} - 1}{2^t} + \mathbf{Adv}_3(\mathcal{A}). \tag{7}$$

$\mathbf{G}_3 \to \mathbf{G}_4$: The next game hop consists of guessing under which KEK the adversary will make its challenge query. That is, in game $\mathbf{G}_4$, the challenger keeps a counter k which tracks the current *key phase*. The counter is initialized to 1 at the start of the game and incremented by one each time the adversary successfully calls oracle ROTKEY and updates the KEK. Additionally, during the setup of the game the challenger guesses the key phase during which the challenge encryption query will occur by drawing an integer $\omega \in \{1, 2, \ldots, m\}$ u.a.r. When the adversary makes its single query to oracle RO$-ENC, the challenger checks if $\omega = k$, i.e. if the guess was correct. If yes, the game continues as in game $\mathbf{G}_3$. If the guess is incorrect, the challenger responds to the challenge query and all future queries by $\perp$. Additionally, if $k = \omega$ and the adversary calls oracles CORR or ROTKEY before oracle RO$-ENC, the triggering query and all following are responded to with $\perp$. Note that such a query indicates that the guess was incorrect, since challenge queries are disallowed after a key compromise in the current key phase. Let E denote the event that the guess is correct, i.e. that $k = \omega$ at the time of the challenge query. Then $\Pr[\mathsf{E}] = \frac{1}{m}$, so for any adversary $\mathcal{A}$,

$$\begin{aligned}\Pr[\mathbf{G}_4(\mathcal{A})] &= \Pr[\mathbf{G}_4(\mathcal{A}) \mid \mathsf{E}] \cdot \Pr[\mathsf{E}] + \Pr[\mathbf{G}_4(\mathcal{A}) \mid \neg\mathsf{E}] \cdot \Pr[\neg\mathsf{E}] \\ &= \frac{1}{m}\left(\Pr[\mathbf{G}_3(\mathcal{A}) \mid \mathsf{E}] + \frac{m-1}{2}\right).\end{aligned}$$

Here $\Pr[\mathbf{G}_4(\mathcal{A}) \mid \mathsf{E}] = \Pr[\mathbf{G}_3(\mathcal{A}) \mid \mathsf{E}]$ since the games are identical when event E occurs. Further, $\Pr[\mathbf{G}_4(\mathcal{A}) \mid \neg\mathsf{E}] = \frac{1}{2}$ since the secret bit b in the game is completely hidden from the adversary when E does not occur, so the output of $\mathcal{A}$ is independent of b. The outcome of game $\mathbf{G}_3$ is independent of event E, so $\Pr[\mathbf{G}_3(\mathcal{A}) \mid \mathsf{E}] = \Pr[\mathbf{G}_3(\mathcal{A})]$. Therefore

$$\begin{aligned}\mathbf{Adv}_4(\mathcal{A}) &:= 2\left|\Pr[\mathbf{G}_4(\mathcal{A}) \Rightarrow \text{true}] - \frac{1}{2}\right| \\ &= 2\left|\frac{1}{m}\left(\Pr[\mathbf{G}_3(\mathcal{A})] + \frac{m-1}{2}\right) - \frac{1}{2}\right| \\ &= \frac{1}{m} \cdot 2\left|\Pr[\mathbf{G}_3(\mathcal{A})] + \frac{m-1}{2} - \frac{m}{2}\right| = \frac{1}{m} \cdot \mathbf{Adv}_3(\mathcal{A}).\end{aligned}$$

This shows that

$$\mathbf{Adv}_3(\mathcal{A}) = m \cdot \mathbf{Adv}_4(\mathcal{A}), \tag{8}$$

for any adversary $\mathcal{A}$.

$\mathbf{G}_4 \to \mathbf{G}_5$: In this game hop, the wrapped DEK in the "real" challenge query is replaced by a random string, so that the headers of the challenge in game $\mathbf{G}_5$ with hidden bit $b = 0$ and with $b = 1$ are distributed identically. That is, game $\mathbf{G}_5$ is identical to $\mathbf{G}_4$ when $b = 0$, but when $b = 1$ the header output in response to the single RO$-ENC query is replaced by a random string of the appropriate length. We bound the difference in success probability by the advantage of an adversary $\mathcal{B}_{\mathsf{pkw}}$ against the find$-1cpa security of PKW. That is, for all adversaries $\mathcal{A}$, we construct $\mathcal{B}$ such that

$$|\Pr[\mathbf{G}_4(\mathcal{A})] - \Pr[\mathbf{G}_5(\mathcal{A})]| \leq \mathbf{Adv}^{\text{find\$-1cpa}}_{\mathsf{PKW}}(\mathcal{B}_{\mathsf{pkw}}).$$

REDUCTION. The reduction works as follows. Adversary $\mathcal{B}_{\mathsf{pkw}}$ acts as the challenger in game $\mathbf{G}_4$, and begins by sampling a bit b' and a key phase guess $\omega \in \{1, \ldots, m\}$ uniformly at random. It also initializes a key phase counter k to 1. It then runs adversary $\mathcal{A}$, simulating access to oracles RO$-ENC, ENC, SHRED, CORR and ROTKEY. If $b' = 0$, adversary $\mathcal{B}_{\mathsf{pkw}}$ simulates all oracles on its own (there is no embedded PKW challenge). To do this, it runs $\mathsf{PKW.KeyGen}()$ to obtain a KEK and uses this to respond to ENC, SHRED, ROTKEY and CORR queries, as would the challenger in game $\mathbf{G}_4$ when $b = 0$ (and $\mathbf{G}_5$, as the two games are identical in this case). The challenge query to oracle RO$-ENC is responded to with random strings, conditioned on it occurring in the guessed key phase ω.

If $b' = 1$, adversary $\mathcal{B}_{\mathsf{pkw}}$ again acts as the challenger in game $\mathbf{G}_4$, except that it uses the NEW-RO$-WRAP oracle in the PKW game to produce the challenge header for $\mathcal{A}$. In each key phase $k < \omega$, adversary $\mathcal{B}_{\mathsf{pkw}}$ runs $\mathsf{PKW.KeyGen}()$ and uses the resulting KEK to simulate access to ENC, SHRED, ROTKEY and CORR for adversary $\mathcal{A}$. When key phase $k = \omega$ is reached, adversary $\mathcal{B}_{\mathsf{pkw}}$ draws a file identifier $id_c \in \{0,1\}^t$ and a DEK K_c u.a.r. and issues query NEW-RO$-WRAP$(id_c, \varepsilon, K_c)$ to its challenger. In response, adversary $\mathcal{B}_{\mathsf{pkw}}$ receives a new KEK which has been punctured on id_c, and a real or random wrap h_c of K_c. The KEK is used to simulate oracles ENC, SHRED and CORR for adversary $\mathcal{A}$. When adversary $\mathcal{A}$ makes the challenge query RO$-ENC$(F)$, adversary $\mathcal{B}_{\mathsf{pkw}}$ uses K_c to encrypt the plaintext and returns id_c, h_c and the file ciphertext to $\mathcal{A}$. In the following key phases ($k > \omega$), the process is repeated until adversary $\mathcal{A}$ shreds the challenge file identifier. That is, until adversary $\mathcal{A}$ issues query SHRED(id_c), adversary $\mathcal{B}_{\mathsf{pkw}}$ responds to a ROTKEY query from $\mathcal{A}$ by issuing a new NEW-RO$-WRAP$(id_c, \varepsilon, K_c)$ query to its challenger, in response to which it receives (sk_k, h'_c). It uses the header h'_c to simulate the update of the challenge header and then uses the new punctured KEK sk_k to simulate the remaining key rotation updates, as well as to respond to all following queries from $\mathcal{A}$. After adversary $\mathcal{A}$ has issued the SHRED(id_c) query, the challenge file no longer needs to be updated, so adversary $\mathcal{B}_{\mathsf{pkw}}$ instead simulates key rotation queries by running $\mathsf{PKW.KeyGen}()$ and uses the resulting fresh KEK to simulate oracles ENC, SHRED and CORR.

Adversary $\mathcal{B}_{\mathsf{pkw}}$ runs until adversary $\mathcal{A}$ halts and outputs bit $b_{\mathcal{A}}$. Then adversary $\mathcal{B}_{\mathsf{pkw}}$ halts and returns 1 if $b_{\mathcal{A}} = b'$, 0 otherwise. Let b_{pkw} denote the secret bit in game $\mathbf{G}^{\text{find\$-1cpa}}_{\mathsf{PKW}}(\mathcal{B}_{\mathsf{pkw}})$. With the strategy described above, adversary $\mathcal{B}_{\mathsf{pkw}}$ simulates game $\mathbf{G}_4$ for $\mathcal{A}$ when $b_{pkw} = 1$ and $\mathbf{G}_5$ when $b_{pkw} = 0$.

SOUNDNESS. In the reduction, DEK-wraps in key phase ω are potentially reordered: the challenge header is produced first, even if the first query from $\mathcal{A}$ in key phase ω is to oracle ENC. For this simulation to be correct despite the potential reordering, we rely on that the PKW scheme is consistent (Definition 9), that file identifiers drawn by $\mathcal{B}_{\mathsf{pkw}}$ when simulating real encryption queries do not collide with id_c, and that id_c does not coincide with a previously shredded file id (as in that case the output of wrap would be undefined). The first is by assumption, the second by exclusion in $\mathbf{G}_3$, and the third by exclusion in $\mathbf{G}_1$. Consistency guarantees that simulated responses to ENC-oracle queries are distributed correctly even if the key used to generate them has been punctured on id_c in the simulation, but not in the original game. The exclusion of file id collisions is necessary since if adversary $\mathcal{B}_{\mathsf{pkw}}$ tries to simulate an ENC-query for $\mathcal{A}$ prior to the challenge query and draws id_c, then the simulation will fail since the KEK has been punctured on id_c due to the pre-computed challenge.

We also need that PKW is puncture invariant (Definition 8) for the simulation to be sound. This ensures that the KEK is correctly distributed at the point of corruption, despite being punctured on the challenge file id first in the simulation. In the simulated game, adversary $\mathcal{A}$ may choose to query oracle SHRED on other file identifiers before shredding id_c. Additionally, a correct guess of the key phase in which the challenge query occurs is needed for the reordering strategy to work, as adversary $\mathcal{B}_{\mathsf{pkw}}$ would not be able to respond to a corruption query in key phase ω with an unpunctured KEK. This issue will not arise if it is guaranteed that the challenge query happens in key phase ω, since adversary $\mathcal{A}$ is then disallowed from querying oracle CORR without first shredding the challenge file id.

BOUND. Let $b_{\mathcal{B}}$ denote the value of the bit output by $\mathcal{B}_{\mathsf{pkw}}$ when it halts. By rewriting the find\$-1cpa advantage from Definition 10 to condition on the value of bit b_{pkw}, we have

$$\mathbf{Adv}^{\text{find\$-1cpa}}_{\mathsf{PKW}}(\mathcal{B}_{\mathsf{pkw}}) = \left|\Pr\left[\, b_{\mathcal{B}} = 1 \,\middle|\, b_{pkw} = 1 \,\right] - \Pr\left[\, b_{\mathcal{B}} = 1 \,\middle|\, b_{pkw} = 0 \,\right]\right|.$$

Given that $b_{\mathcal{B}} = 1$ precisely when $b_{\mathcal{A}} = b$, we see that

$$\Pr\left[\, b_{\mathcal{B}} = 1 \,\middle|\, b_{pkw} = 1 \,\right] = \Pr\left[\, b_{\mathcal{A}} = b \,\middle|\, b_{pkw} = 1 \,\right] = \Pr\left[\, \mathbf{G}_4(\mathcal{A}) \,\right]$$

and

$$\Pr\left[\, b_{\mathcal{B}} = 1 \,\middle|\, b_{pkw} = 0 \,\right] = \Pr\left[\, b_{\mathcal{A}} = b \,\middle|\, b_{pkw} = 0 \,\right] = \Pr\left[\, \mathbf{G}_5(\mathcal{A}) \,\right].$$

Hence $\mathbf{Adv}^{\text{find\$-1cpa}}_{\mathsf{PKW}}(\mathcal{B}_{\mathsf{pkw}}) = |\Pr[\,\mathbf{G}_4(\mathcal{A})\,] - \Pr[\,\mathbf{G}_5(\mathcal{A})\,]|$. In conclusion, we have shown that for each adversary $\mathcal{A}$ there exists an adversary $\mathcal{B}_{\mathsf{pkw}}$ such that

$$\mathbf{Adv}_4(\mathcal{A}) \leq 2 \cdot \mathbf{Adv}^{\text{find\$-1cpa}}_{\mathsf{PKW}}(\mathcal{B}_{\mathsf{pkw}}) + \mathbf{Adv}_5(\mathcal{A}). \tag{9}$$

Adversary $\mathcal{B}_{\mathsf{pkw}}$ makes at most m queries to oracle NEW-RO\$-WRAP, where m is the maximum number of KEKs used in the simulation, generated by at most $m-1$ key rotation queries by adversary $\mathcal{A}$.

$\mathbf{G}_5 \to \mathbf{G}_6$: The final step of the proof consists of replacing the file ciphertexts in the real world by uniformly random strings. Let $\mathbf{G}_6$ be identical to game $\mathbf{G}_5$, except that in response to a challenge query RO\$-ENC(F), the file ciphertext is drawn u.a.r. from $\{0,1\}^{\mathsf{cl}(|F|)}$, regardless of the value of the secret bit b. We bound any difference of advantage due to the change through a reduction to the ind\$-cpa security of the AEAD scheme. To this end, we design an adversary $\mathcal{B}_{\mathsf{aead}}$, such that for all adversaries $\mathcal{A}$ it holds that $\mathbf{Adv}_5(\mathcal{A}) \leq 2 \cdot \mathbf{Adv}^{\text{ind\$-cpa}}_{\mathsf{AEAD}}(\mathcal{B}_{\mathsf{aead}}) + \mathbf{Adv}_6(\mathcal{A})$.

The reduction is straightforward and works as follows. Adversary $\mathcal{B}_{\mathsf{aead}}$ acts as the challenger in game $\mathbf{G}_5$ and begins by drawing $b' \leftarrow\!\!\$\ \{0,1\}$ and running $\mathsf{PKW.KeyGen}()$ to generate an initial KEK, which it uses to simulate responses to queries from $\mathcal{A}$ to oracles ENC, SHRED, CORR and ROTKEY. When adversary $\mathcal{A}$ makes the challenge query RO\$-ENC(F), adversary $\mathcal{B}_{\mathsf{aead}}$ queries oracle NEW in game $\mathbf{G}^{\text{ind\$-cpa}}_{\mathsf{AEAD}}$ to generate a new DEK. Additionally it samples $id \leftarrow\!\!\$\ \{0,1\}^t$, $h \leftarrow\!\!\$\ \{0,1\}^{\mathsf{PKW.cl}(t)}$ and $N \leftarrow\!\!\$\ \{0,1\}^n$. Adversary $\mathcal{B}_{\mathsf{aead}}$ then issues query $C \leftarrow \text{RO\$}(1, N, \varepsilon, F)$ to its challenge oracle and sets $C_1 \leftarrow N \| C$. It draws $C_0 \leftarrow\!\!\$\ \{0,1\}^{\mathsf{cl}(|F|)}$ and returns $(id, h, C_{b'})$ to adversary $\mathcal{A}$. When adversary $\mathcal{A}$ halts and returns a bit b^*, adversary $\mathcal{B}_{\mathsf{aead}}$ also halts and returns 1 if $b^* = b'$, otherwise 0.

Let b denote the secret bit in game $\mathbf{G}^{\text{ind\$-cpa}}_{\mathsf{AEAD}}$ played by $\mathcal{B}_{\mathsf{aead}}$. Adversary $\mathcal{B}_{\mathsf{aead}}$ perfectly simulates game $\mathbf{G}_5$ for $\mathcal{A}$ when $b = 1$ and $\mathbf{G}_6$ when $b = 0$. Hence, for all adversaries $\mathcal{A}$ making exactly one query to oracle RO\$-ENC in game $\mathbf{G}_5$, we have shown that there exists an adversary $\mathcal{B}_{\mathsf{aead}}$, making a single NEW- and RO\$-query, such that

$$\mathbf{Adv}_5(\mathcal{A}) \leq 2 \cdot \mathbf{Adv}^{\text{ind\$-cpa}}_{\mathsf{AEAD}}(\mathcal{B}_{\mathsf{aead}}) + \mathbf{Adv}_6(\mathcal{A}). \tag{10}$$

$\mathbf{G}_6$: In game $\mathbf{G}_6$, the response to the single challenge query (as well as to all other oracle queries) has the same distribution when $b = 1$ as when $b = 0$. Therefore

$$\mathbf{Adv}_6(\mathcal{A}) = 0. \tag{11}$$

Together, equations (5)-(11) give the theorem statement. This concludes the proof. □

E.2 Proof of Theorem 8: PFS[PKW, AEAD] is find\$-rcpa, via PKW find\$-rcpa

The proof proceeds by a series of game hops, starting with game $\mathbf{G}_0$, which is equivalent to $\mathbf{G}^{\text{find\$-rcpa}}_{\mathsf{PFS[PKW,AEAD]}}$, and ending in game $\mathbf{G}_4$, where the distribution of the output components of the challenge oracle RO\$-ENC is independent of the challenge bit b. We define $\mathbf{Adv}_i(\mathcal{A}) := 2\left|\Pr[\,\mathbf{G}_i(\mathcal{A}) \Rightarrow \text{true}\,] - \frac{1}{2}\right|$ for $i \in \{0, \ldots, 4\}$ for any adversary $\mathcal{A}$.

$\mathbf{G}_0 \to \mathbf{G}_1$: In the first game hop, file identifiers for challenge encryptions in the real and ideal world are prevented from being drawn from the set of already shredded identifiers, letting RO\$-ENC and all subsequent oracles return $\perp$ in such case. This step is identical to the first game hop in the proof of Theorem 7. Thus, for all adversaries $\mathcal{A}$ it holds that

$$\mathbf{Adv}_0(\mathcal{A}) \leq 4q_{ro\$} \cdot \frac{q_s}{2^t} + \mathbf{Adv}_1(\mathcal{A}), \tag{12}$$

We refer the reader to the proof of Theorem 7 in Appendix E.1 for details.

$\mathbf{G}_1 \to \mathbf{G}_2$: In the second game hop, we remove the need to handle encryption and challenge queries with coinciding file identifiers. Let $\mathcal{S}_{id}$ be the set of all file identifiers output by oracle ENC or RO\$-ENC which have not been shredded in an earlier key phase. (That is, $\mathcal{S}_{id}$ includes file identifiers shredded in the current key phase, but not those for which at least one key rotation has taken place after the shredding.) Let game $\mathbf{G}_2$ be identical to game $\mathbf{G}_1$, except that if during an encryption or challenge query, a file

identifier is drawn which coincides with an id in $\mathcal{S}_{id}$, then a flag bad_2 is set to true and the query is responded to with $\perp$.

The probability of bad_2 occurring is hence upper bounded by the collision probability when drawing $q_e + q_{ro\$}$ elements from a set of size at most $q_e + q_{ro\$}$. Hence, by the birthday bound, $\Pr[\mathsf{bad}_2] \leq \frac{1}{2} \cdot \frac{(q_e+q_{ro\$})^2}{2^t}$. Since games $\mathbf{G}_2$ and $\mathbf{G}_1$ are identical-until-bad, the fundamental lemma of game playing [7] gives $|\Pr[\mathbf{G}_1] - \Pr[\mathbf{G}_2]| \leq \Pr[\mathsf{bad}_2] \leq \frac{(q_e+q_{ro\$})^2}{2^{t+1}}$, which means that

$$\mathbf{Adv}_1(\mathcal{A}) \leq 2 \cdot \frac{(q_e + q_{ro\$})^2}{2^{t+1}} + \mathbf{Adv}_2(\mathcal{A}). \tag{13}$$

$\mathbf{G}_2 \rightarrow \mathbf{G}_3$: In this game hop, the distribution of the headers output by oracle RO$-ENC is made independent of the value of the secret bit b. To this end, the headers generated by oracle RO$-ENC when $b = 1$ are replaced by random strings in game $\mathbf{G}_3$. We bound the advantage difference of adversary $\mathcal{A}$ resulting from this change by the advantage of an adversary $\mathcal{B}_{\mathsf{pkw}}$ against the find$-rcpa security of PKW (Definition 10).

Adversary $\mathcal{B}_{\mathsf{pkw}}$ acts as the challenger in game $\mathbf{G}_2$, drawing a secret bit b' at the start of the game and initializing a counter u to keep track of the key phase (the counter is incremented upon successful key rotation queries from $\mathcal{A}$). To generate the initial PKW key, adversary $\mathcal{B}_{\mathsf{pkw}}$ uses oracle NEW. To simulate the header part of the response to an ENC- or RO$-ENC-query in the real world (when the secret bit $b' = 1$), $\mathcal{B}_{\mathsf{pkw}}$ issues query $h \leftarrow \textsc{Wrap}(u, id, \varepsilon, K)$ or $h \leftarrow \textsc{Ro\$-Wrap}(u, id, \varepsilon, K)$, respectively, for a freshly sampled AEAD key K and file identifier id. Additionally, adversary $\mathcal{B}_{\mathsf{pkw}}$ keeps a table $\mathsf{T}[\cdot,\cdot]$ in which it stores K id and h. The table is used to simulate queries to oracle ROTKEY, where adversary $\mathcal{B}_{\mathsf{pkw}}$—after calling oracle NEW to initialize a new PKW key—uses oracles WRAP and RO$-WRAP to re-wrap the DEKs in the table under the new key in order to generate the new headers. The table is updated accordingly. Queries to oracles SHRED and CORR from adversary $\mathcal{A}$ are relayed to oracles PUNC and CORR, respectively. A SHRED query additionally deletes any table entry corresponding to the shredded file id. When adversary $\mathcal{A}$ halts and outputs its bit guess b^*, $\mathcal{B}_{\mathsf{pkw}}$ halts and outputs 1 if $b^* = b'$, otherwise 0.

Thanks to the hop to game $\mathbf{G}_2$ where the need to handle coinciding file identifiers was removed, adversary $\mathcal{B}_{\mathsf{pkw}}$ is guaranteed to be tag-respecting. Let b denote the secret bit drawn by the challenger in the PKW find$-rcpa game played by $\mathcal{B}_{\mathsf{pkw}}$. With the strategy described above, adversary $\mathcal{B}_{\mathsf{pkw}}$ simulates game $\mathbf{G}_2$ for adversary $\mathcal{A}$ when $b = 1$ and $\mathbf{G}_3$ when $b = 0$. Hence

$$\mathbf{Adv}_2(\mathcal{A}) \leq 2 \cdot \mathbf{Adv}^{\text{find\$-rcpa}}_{\mathsf{PKW}}(\mathcal{B}_{\mathsf{pkw}}) + \mathbf{Adv}_3(\mathcal{A}). \tag{14}$$

$\mathbf{G}_3 \rightarrow \mathbf{G}_4$: In this final game hop, the AEAD encryption C of the file plaintext in the real world is replaced by a random string of the appropriate length. That is, game $\mathbf{G}_4$ is identical to game $\mathbf{G}_3$, except that in response to a challenge query RO$-ENC($F$), the challenger draws $C \leftarrow\!\!\$\ \{0,1\}^{\mathsf{AEAD.cl}(|F|)}$ and sets $C_1 \leftarrow N\|C$, where N is a freshly sampled AEAD nonce. As before, C_0 is drawn u.a.r. from $\{0,1\}^{\mathsf{PFS.cl}(|F|)}$ and the challenger returns (id, h, C_b) to the adversary, where b is the secret bit and id and h is a freshly sampled file identifier and header, respectively.

We bound $|\Pr[\mathbf{G}_3] - \Pr[\mathbf{G}_4]$ via a reduction to the ind$-cpa security of AEAD, as follows. Let $\mathcal{B}_{\mathsf{aead}}$ be an adversary against the ind$-cpa security of AEAD. $\mathcal{B}_{\mathsf{aead}}$ simulates game $\mathbf{G}_3$ for adversary $\mathcal{A}$, drawing a secret bit b' and acting as the challenger in the game, with the following exceptions. To respond to a challenge query RO$-ENC($F$), adversary $\mathcal{B}_{\mathsf{aead}}$ calls oracle NEW to initialize a new AEAD key, as well as increments a key counter u. It then draws a file identifier $id \leftarrow\!\!\$\ \{0,1\}^t$ and checks if id has been shredded in the current key phase, i.e., if adversary $\mathcal{A}$ has issued query SHRED(id) since the last key rotation. If the id has been shredded, $\mathcal{B}_{\mathsf{aead}}$ returns $\perp$ to $\mathcal{A}$ now and in all subsequent queriey, in accordance with game $\mathbf{G}_1$. Otherwise $\mathcal{B}_{\mathsf{aead}}$ samples a random header $h \leftarrow \mathcal{H}$ and a file nonce $N \leftarrow\!\!\$\ \{0,1\}^n$. It then queries its challenge oracle, $C \leftarrow \textsc{Ro\$}(u, N, \varepsilon, F)$, sets $C_1 \leftarrow N\|C$, samples $C_0 \leftarrow\!\!\$\ \{0,1\}^{\mathsf{PFS.cl}(|F|)}$ and returns $(id, h, C_{b'})$ to adversary $\mathcal{A}$. When $\mathcal{A}$ halts and outputs a bit b^*, adversary $\mathcal{B}_{\mathsf{aead}}$ halts and returns 1 of $b^* = b'$, 0 otherwise.

Since each query to oracle RO$ is under a new AEAD key, $\mathcal{B}_{\mathsf{aead}}$ is guaranteed to be nonce-respecting. Let b denote the secret bit in game $\mathbf{G}^{\text{ind\$-cpa}}_{\mathsf{AEAD}}$ played by adversary $\mathcal{B}_{\mathsf{aead}}$. Adversary $\mathcal{B}_{\mathsf{aead}}$ simulates game $\mathbf{G}_3$ for $\mathcal{A}$ when $b = 1$ and game $\mathbf{G}_4$ when $b = 0$. Hence $|\Pr[\mathbf{G}_3] - \Pr[\mathbf{G}_4]| \leq \mathbf{Adv}^{\text{ind\$-cpa}}_{\mathsf{AEAD}}(\mathcal{B}_{\mathsf{aead}})$, giving

$$\mathbf{Adv}_3(\mathcal{A}) \leq 2 \cdot \mathbf{Adv}^{\text{ind\$-cpa}}_{\mathsf{AEAD}}(\mathcal{B}_{\mathsf{aead}}) + \mathbf{Adv}_4(\mathcal{A}). \tag{15}$$

$\mathbf{G}_4$: In game $\mathbf{G}_4$, the output of all oracles has the same distribution when $b = 1$ as when $b = 0$. In particular, we note that this holds for the file ciphertext C generated in response to a query RO$-ENC$(F)$, since when $b = 1$, $C = N\|C$ where N and C are randomly sampled from $\{0,1\}^n$ and $\{0,1\}^{\mathsf{AEAD.cl}(|F|)}$, respectively, and when $b = 0$ then C is drawn u.a.r. from $\{0,1\}^{\mathsf{PFS.cl}(|F|)}$, where $\mathsf{PFS.cl}(|F|) = n + \mathsf{AEAD.cl}(|F|)$ by construction of $\mathsf{PFS}[\mathsf{PKW}, \mathsf{AEAD}]$. Hence for all adversaries $\mathcal{A}$ it must hold that

$$\mathbf{Adv}_4(\mathcal{A}) = 0. \tag{16}$$

Combining Equations (12)-(16) gives the theorem bound. □

E.3 Proof of Theorem 9: $\mathsf{PFS}[\mathsf{PKW}, \mathsf{AEAD}]$ is int-ctxt

The proof consists of three game hops, starting with $\mathbf{G}_0$ which is identical to $\mathbf{G}^{\text{int-ctxt}}_{\mathsf{PFS}[\mathsf{PKW},\mathsf{AEAD}]}$ and ending in game $\mathbf{G}_3$. We bound the advantage difference in the first hop by a birthday bound collision probability and in the subsequent hops by the advantage of an adversary against the security of the underlying puncturable key wrapping scheme PKW. The advantage of $\mathcal{A}$ in the final game $\mathbf{G}_3$ is bounded by a reduction to the int-ctxt security of AEAD.

$\mathbf{G}_0 \to \mathbf{G}_1$: The first game hop ensures that the file identifiers output by oracle ENC are unique. That is, game $\mathbf{G}_1$ is identical to game $\mathbf{G}_0$, except that if in an encryption query a previously used file identifier is drawn, then the query is responded to by $\perp$. Let bad_1 be the event that a collision of file identifiers occurs. By the birthday bound, $\Pr[\,\mathsf{bad}_1\,] \leq \frac{q_e^2}{2^{t+1}}$, since there are at most q_e file identifiers sampled during the course of the game, and they are all drawn uniformly at random from a set of size 2^t.

Games $\mathbf{G}_0$ and $\mathbf{G}_1$ are identical-until-bad, hence

$$|\Pr[\,\mathbf{G}_0\,] - \Pr[\,\mathbf{G}_1\,]| \leq \Pr[\,\mathsf{bad}_1\,] \leq \frac{q_e^2}{2^{t+1}} \tag{17}$$

by the fundamental lemma of game playing [7].

$\mathbf{G}_1 \to \mathbf{G}_2$: In the second game hop, forged headers are ruled out by PKW int-ctxt security (Definition 11). This limits the data encryption keys used in the game to those wrapped by the PKW scheme. Let bad_2 be the event that adversary $\mathcal{A}$ successfully forges a header. That is, let $\mathcal{S}_{pkw}$ contain all pairs (id, h) of file identifiers and headers which have been output by the encryption oracle during the current key phase, or which where output by oracle ROTKEY to initiate the key phase, and for which id has not subsequently been shredded. Then we say that bad_2 occurs if adversary $\mathcal{A}$ issues either (1) a query DEC(id, h, C) such that the resulting plaintext is not $\perp$, and $(id, h) \notin \mathcal{S}_{pkw}$, or (2) a query to oracle ROTKEY in which the key is successfully rotated and the input to the oracle includes a file id, header pair $(id, h) \notin \mathcal{S}_{pkw}$.

Let game $\mathbf{G}_2$ be identical to $\mathbf{G}_1$ unless the bad event occurs. If bad_2 happens, the query is aborted and responded to by $\perp$. By the fundamental lemma of game playing [7], we have $\Pr[\,\mathbf{G}_1\,] - \Pr[\,\mathbf{G}_2\,] \leq \Pr[\,\mathsf{bad}\,]$. We bound the probability of bad_2 by the advantage of an adversary $\mathcal{B}_{\mathsf{pkw}}$.

Adversary $\mathcal{B}_{\mathsf{pkw}}$ simulates game $\mathbf{G}_1$ for adversary $\mathcal{A}$, acting as the challenger. It initiates the simulation by calling oracle NEW and setting a counter u to 1. To wrap a DEK in an encryption query, adversary $\mathcal{B}_{\mathsf{pkw}}$ uses its wrapping oracle. To unwrap a header in a decryption query, it uses its unwrapping oracle, and to shred a file identifier it calls the puncturing oracle. In all such queries, adversary $\mathcal{B}_{\mathsf{pkw}}$ uses the key counter u to indicate under which PKW key the query should be processed. To simulate a key rotation query for $\mathcal{A}$, adversary $\mathcal{B}_{\mathsf{pkw}}$ calls oracle NEW to initialize a new PKW key. For each pair (id, h) in the query it then issues query UNWRAP(u, id, ε, h) to unwrap the DEK and then rewraps it under key $u+1$ using oracle WRAP with the id as tag. If all rewraps are successful, it updates $u \leftarrow u+1$.

With this strategy, adversary $\mathcal{B}_{\mathsf{pkw}}$ perfectly simulates game $\mathbf{G}_1$ for adversary $\mathcal{A}$. Additionally, if event bad_2 occurs, $\mathcal{B}_{\mathsf{pkw}}$ wins game $\mathbf{G}^{\text{int-ctxt}}_{\mathsf{PKW}}$. To see this, consider each of the two cases when the bad flag gets set. In case (1), adversary $\mathcal{A}$ makes a query DEC(id, h, C) such that the resulting plaintext is not $\perp$, which means that the header must have unwrapped successfully. (Otherwise the AEAD decryption would fail since there would be no DEK to decrypt with.) Furthermore, the requirement that $(id, h) \notin \mathcal{S}_{pkw}$ ensures that the unwrap query issued by $\mathcal{B}_{\mathsf{pkw}}$ during the simulation of the decryption query counts as a valid forgery. The exclusion of colliding file identifiers in the prior game hop ensures that adversary

$\mathcal{B}_{\mathsf{pkw}}$ is tag-respecting. For the same reasons case (2) corresponds to a valid forgery by $\mathcal{B}_{\mathsf{pkw}}$ in the PKW integrity game. Hence

$$|\Pr[\mathbf{G}_1] - \Pr[\mathbf{G}_2]| \leq \Pr[\mathsf{bad}] \leq \mathbf{Adv}^{\text{int-ctxt}}_{\mathsf{PKW}}(\mathcal{B}_{\mathsf{pkw}}). \tag{18}$$

$\mathbf{G}_2 \to \mathbf{G}_3$: In this game hop, all headers generated in encryption and key rotation queries are replaced by random strings from the header space. Any advantage difference due to the change is bound by a reduction to the find$-cpa security of PKW (Definition 10).

Let game $\mathbf{G}_3$ be identical to $\mathbf{G}_2$, except that in encryption and key rotation queries new headers are not generated by the wrapping algorithm of PKW, but rather drawn uniformly at random from $\{0,1\}^{\mathsf{PKW.cl}(k)}$. Here k is the bit length of the DEKs in the construction. We construct an adversary $\mathcal{C}_{\mathsf{pkw}}$ such that

$$|\Pr[\mathbf{G}_2] - \Pr[\mathbf{G}_3]| \leq \mathbf{Adv}^{\text{find\$-cpa}}_{\mathsf{PKW}}(\mathcal{C}_{\mathsf{pkw}}). \tag{19}$$

Adversary $\mathcal{C}_{\mathsf{pkw}}$ simulates game $\mathbf{G}_2$ for $\mathcal{A}$, acting as the challenger, except that to generate headers for encryption and key rotation queries it uses oracle RO$-WRAP on input a file identifier id and key K. To simulate the unwrapping of headers in decryption and key rotation queries, $\mathcal{C}_{\mathsf{pkw}}$ keeps a table $\mathsf{T}[\cdot,\cdot]$ where it stores the DEKs it has wrapped under the header it received from oracle RO$-WRAP and the file id. That is, after each query $h \leftarrow$ RO$-WRAP$(u, id, \varepsilon, K)$, where u is a counter keeping track of the current key phase, adversary $\mathcal{C}_{\mathsf{pkw}}$ sets $\mathsf{T}[id, h] \leftarrow K$. The table entry is deleted if adversary $\mathcal{A}$ calls oracle SHRED on id. Additionally all table entries are reset after a key rotation query such that only DEKs corresponding to updated headers are kept, and they are stored under the new header.

Queries to oracle SHRED are relayed to oracle PUNC with key index u. To simulate key rotation, adversary $\mathcal{C}_{\mathsf{pkw}}$ calls oracle NEW to initialize a new key and lets $u \leftarrow u + 1$. When adversary $\mathcal{A}$ halts, adversary $\mathcal{C}_{\mathsf{pkw}}$ checks if the win flag has been set to true. If yes, it halts and returns 1, otherwise 0.

Because PKW.Unwrap is a deterministic algorithm, and since (thanks to the previous game hop) only pairs (id, h) which have been part of the response to an encryption query need to be handled by adversary $\mathcal{C}_{\mathsf{pkw}}$ in decryption and key rotation queries, any DEK which would be output by PKW.Unwrap in game $\mathbf{G}_2$ will be stored in the table kept by $\mathcal{C}_{\mathsf{pkw}}$. Additionally adversary $\mathcal{C}_{\mathsf{pkw}}$ is guaranteed to be tag-respecting thanks to the exclusion of event bad_1 in the hop to game $\mathbf{G}_1$. This ensures that the simulation is sound. When the secret bit b in game $\mathbf{G}^{\text{find\$-cpa}}_{\mathsf{PKW}}$ is 1, adversary $\mathcal{C}_{\mathsf{pkw}}$ simulates game $\mathbf{G}_2$ for $\mathcal{A}$. When $b = 0$ the simulation corresponds to game $\mathbf{G}_3$. Let $b^*_{\mathcal{C}}$ denote the bit returned by adversary $\mathcal{C}_{\mathsf{pkw}}$. Then

$$\begin{aligned}
\mathbf{Adv}^{\text{find\$-cpa}}_{\mathsf{PKW}}(\mathcal{C}_{\mathsf{pkw}}) &= 2\left|\Pr\left[\,\mathbf{G}^{\text{find\$-cpa}}_{\mathsf{PKW}}(\mathcal{C}_{\mathsf{pkw}}) \Rightarrow \text{true}\,\right] - \frac{1}{2}\right| \\
&= \left|\Pr\left[\,b^*_{\mathcal{C}} = 1 \,\middle|\, b = 1\,\right] - \Pr\left[\,b^*_{\mathcal{C}} = 1 \,\middle|\, b = 0\,\right]\right| \\
&= |\Pr[\mathbf{G}_2(\mathcal{A})] - \Pr[\mathbf{G}_3(\mathcal{A})]|\,,
\end{aligned}$$

which establishes Equation (19).

$\mathbf{G}_3$: Finally, we show that the advantage of adversary $\mathcal{A}$ in game $\mathbf{G}_3$ is bounded by that advantage of an adversary $\mathcal{B}_{\mathsf{aead}}$ against the int-ctxt security of AEAD (Definition 3). Adversary $\mathcal{B}_{\mathsf{aead}}$ simulates game $\mathbf{G}_3$ for $\mathcal{A}$, using oracle NEW to initiate a new AEAD key each time a new DEK would be drawn in the game, that is, in each encryption query where a file id id is drawn which does not coincide with a previously used id and which has not been shredded. To keep track of the initiated keys, adversary $\mathcal{B}_{\mathsf{aead}}$ keeps a key counter j which is incremented each time a call to oracle NEW is made. It additionally keeps a table $\mathsf{T}[\cdot,\cdot]$ in which it stores the key indices corresponding to a certain pair (id, h). To simulate a file encryption for $\mathcal{A}$, adversary $\mathcal{B}_{\mathsf{aead}}$ calls oracle ENC in game $\mathbf{G}^{\text{int-ctxt}}_{\mathsf{AEAD}}$ under the index of the new key. To simulate query DEC$(id, h, N\|C)$ from $\mathcal{A}$, adversary $\mathcal{B}_{\mathsf{aead}}$ retrieves the key index $j \leftarrow \mathsf{T}[id, h]$ and issues query DEC(j, N, ε, C) to its decryption oracle. To simulate shredding of file identifier id, $\mathcal{B}_{\mathsf{aead}}$ sets $\mathsf{T}[id, *]$ to $\perp$ and additionally adds id to a set $\mathcal{S}_{\mathcal{P}}$ such that it can keep track of the shredded identifiers (and avoid providing a simulated encryption on a shredded id). When $\mathcal{A}$ makes a key rotation query, adversary $\mathcal{B}_{\mathsf{aead}}$ first checks that for all file identifier and header pairs (id_i, h_i) in the input, $\mathsf{T}[id_i, h_i] \neq \perp$. If the check passes, $\mathcal{B}_{\mathsf{aead}}$ samples a new random header h'_i for each id_i and updates the table by setting $\mathsf{T}[id_i, h'_i] \leftarrow \mathsf{T}[id_i, h_i]$, transferring the key index to the new file identifier and header pair. It additionally deletes all old entries in the table and sets $\mathcal{S}_{\mathcal{P}} \leftarrow \emptyset$.

This way, adversary $\mathcal{B}_{\mathsf{aead}}$ perfectly simulates game $\mathbf{G}_3$ for adversary $\mathcal{A}$. Furthermore, $\Pr[\mathbf{G}_3(\mathcal{A})] \leq \Pr[\mathbf{G}^{\text{int-ctxt}}_{\mathsf{AEAD}}(\mathcal{B}_{\mathsf{aead}})]$, since if $\mathcal{A}$ wins in game $\mathbf{G}_3$, it means that it has submitted a query $\textsc{Dec}(id, h, N\|C)$ such that the result is not $\perp$ and $(id, h, N\|C) \notin \mathcal{S}$, where $\mathcal{S}$ is defined in as in the PFS integrity game in Figure 12. To simulate the query, $\mathcal{B}_{\mathsf{aead}}$ makes query $\textsc{Dec}(\mathsf{T}[id, h], N, \varepsilon, C)$ and since the decryption is successful for $\mathcal{A}$, we know directly that it was also successful for $\mathcal{B}_{\mathsf{aead}}$ (meaning that the resulting plaintext is not $\perp$). Hence it only remains to show that the query counts as a valid forgery. I.e., that $\mathcal{B}_{\mathsf{aead}}$ has not previously made a decryption query on $(\mathsf{T}[id, h], N, \varepsilon, C)$. From $(id, h, N\|C) \notin \mathcal{S}$ we know that at least one of the components is new[12], or that the combination (i.e. the tuple taken as a whole) is new, or that id has been shredded on. We consider each case separately.

1. id has been shredded: By construction of $\mathcal{B}_{\mathsf{aead}}$ this implies that $\mathsf{T}[id, h] = \perp$, contradicting the fact that the result of the query was not $\perp$. Hence this case is ruled out.
2. (id, h) is new: This case is ruled out by the hop to game $\mathbf{G}_2$. (It corresponds to $\mathsf{T}[id, h] = \perp$ in the simulation.)
3. N or C is new: Then $(\mathsf{T}[id, h], N, \varepsilon, C)$ is new, so the decryption query by $\mathcal{B}_{\mathsf{aead}}$ is a valid AEAD forgery.
4. The combination of id, h and $N\|C$ is new (meaning each component may have appeared separately in previous outputs from $\textsc{Enc}$, but not the entire tuple together): Then the tuple $(\mathsf{T}[id, h], N, \varepsilon, C)$ is new (since there are no duplicates in the key index table T) leading to a valid forgery by $\mathcal{B}_{\mathsf{aead}}$.

This shows that

$$\Pr[\mathbf{G}_3] \leq \mathbf{Adv}^{\text{int-ctxt}}_{\mathsf{AEAD}}(\mathcal{B}_{\mathsf{aead}}), \tag{20}$$

where adversary $\mathcal{B}_{\mathsf{aead}}$ makes at most q_e calls to oracle $\textsc{New}$, q_e calls to oracle $\textsc{Enc}$ and q_d calls to oracle $\textsc{Dec}$ in game $\mathbf{G}^{\text{int-ctxt}}_{\mathsf{AEAD}}$.

Combining Equations (17)-(20) yields the claimed bound on the advantage of $\mathcal{A}$. □

F Puncturable Key Rap

Kien Tuong Truong kindly provided this follow-up to the key rap by Rogaway and Shrimpton [51].

If you want to store all your files on the cloud
And you ask yourself: "Now, is there somehow
I can erase my files securely in the case
Of a guy that can steal long-term-keys?" Just you wait!

Just sample a key that you'll use for your files
Then take a new one which you'll keep for a while
The first one's a DEK which we will be encrypting
By using the KEK, and you got "key wrapping"

But that's not enough! And that's why I'm rapping
To show you what happens if you puncture key wrapping
Just update your key and then you're all set
And leaves you an old key that you can forget

With puncturable key wrapping
You know that I'm happier now
Because it's almost 2023
You need some forward secrecy!

If you are annoyed with one RTT
Then send early data, and that by design
Allows you to be quick, but there is a price
Not forward secure, that's not really nice

But here comes our new scheme, to rescue you out
The server's key is punctured, and that leaves no doubt
Don't worry you really don't have to trust me
We managed to prove this in MSKE

With puncturable key wrapping
You know that I'm happier now
Because it's almost 2023
You need some forward secrecy!

With puncturable key wrapping
You know that I'm happier now
Because it's almost 2023
You need some forward secrecy!

[12] New here means "not part of the output of the encryption oracle in response to an earlier query".