Adaptive Security for Constrained PRFs

Kaishuo Cheng and Joseph Jaeger

School of Cybersecurity and Privacy
Georgia Institute of Technology
Atlanta, Georgia, USA
{kcheng89, josephjaeger}@gatech.edu
https://cc.gatech.edu/~kcheng89
https://cc.gatech.edu/~josephjaeger

Abstract. There is a gap between the security of constrained PRFs required in some applications and the security provided by existing definitions. This gap is typically patched by only considering nonadaptive security or manually mixing the CPRF with a random oracle (implicitly constructing a new CPRF) to achieve adaptive security. We fill this gap with a new definition for constrained PRFs with strong adaptive security properties and proofs that it is achieved by practical constructions based on the cascade PRF (which generalizes the GGM construction) and AMAC. We apply the definition for analyzing searchable symmetric encryption and puncturable key wrapping.

Contents

1 Introduction . . . 2
2 Preliminaries . . . 4
3 (Constrained) PRFs: Old and New Definitions . . . 6
3.1 Function Families . . . 6
3.2 Constrained Function Families . . . 7
3.3 Simple Security Claims . . . 9
4 Applications of SIM*-AC-CPRF Security . . . 10
4.1 Searchable Symmetric Encryption . . . 10
4.2 Puncturable Key Wrapping . . . 11
5 Adaptively Secure Constructions . . . 12
5.1 Hashed Constrained PRF(s) . . . 12
5.2 Composed Construction . . . 14
5.3 Cascade-based Constructions . . . 16
5.4 Augmented MAC . . . 18
A Proof of Theorem 1 (SIM*-AC-CPRF implies IND*-AC-CPRF) . . . 21
B Searchable Symmetric Encryption Application . . . 21
C Puncturable Key Wrapping Application . . . 28
D Proofs for Hashed Constrained PRF(s) . . . 34
E Proof of Theorem 4 (Security of Composed Construction) . . . 41
F Proof for Theorem 5 (Security of Cascade-based Constructions) . . . 47
G SIM*-AC-PRF Security With Leakage . . . 49

A preliminary version of this work appears in the Crypto 2025 proceedings [15], ©IACR 2025.
This is the full version.

1 Introduction

Constrained PRFs [9,12,25] are an advanced form of PRF that allows for delegation of the ability to evaluate the PRF on a subset of the domain. They have proven useful in a wide variety of settings including forward secure searchable encryption [10,11,17,25,30,31,33,34], encrypted file storage [2], session resumption for key exchange protocols [1,2], constructions using indistinguishable obfuscation [29], and vector commitments [3,4].

We argue that there is gap between the level of adaptivity provided by the standard security definitions for constrained PRFs and what is required of some applications. In some senses, the attacker is forced to commit to decisions about the nature of its queries before it "should" have to. This restriction hinders the analysis of protocols built based on constrained PRFs that require greater flexibility. We fill this gap with a new, strongly adaptive definition of security for constrained PRFs and show that this definition suffices for analyzing a forward secure searchable encryption scheme (SSE) [11] and a puncturable key wrapping scheme (PKW) [2] whose prior analyses ran into this gap. They bridged the gap, respectively, by manually mixing the constrained PRF with a random oracle or leaving the analysis to future work. We analyze the AMAC [5], cascade [6], and Goldreich-Goldwasser-Micali (GGM) [19] PRFs as constrained PRFs, giving modular proofs that they achieve our definition under appropriate idealization of their low-level components.

1.1 Our Contributions

Constrained PRF definitions. A normal PRF computes a function $\mathsf{F}(K, x) = y$ so that y should look random from the perspective of an attacker if K is uniformly random. A constrained PRF (CPRF) adds an extra functionality $\mathsf{F.Con}(K, R)$ that outputs a constrained key K_R which can be used to compute $\mathsf{F}(K, x)$ for any $x \in R$. Intuitively, security requires that challenge $F(K, x)$ values be indistinguishable from random even if the attacker learns constrained keys K_R, as long as $x \in R$ never holds. The standard security definition explicitly disallows making queries that would break this last condition. It is adaptive if the attacker can arbitrarily interweave its requests for challenge values and constrained keys and nonadaptive if all challenge requests must be made before constrain request. We call these IND*-AC-CPRF (adaptive) and IND*-NC-CPRF (non-adaptive) security respectively.

However, even IND*-AC-CPRF security is not adaptive enough for some applications. Its strict restriction on overlapping challenge and constrain queries implicitly embeds a measure of non-adaptivity because the first time an adversary wants to access $\mathsf{F}(K, x)$ it must commit to whether this will be a challenge value (accessed by asking for the challenge) or constrained value (accessed by asking for the constrained key $K_{\{x\}}$). In some senses, it philosophically captures that $\mathsf{F}(K, x)$ should look random if x is never constrained, but does not directly capture that $\mathsf{F}(K, x)$ should look random *until* x is constrained. This observation is not merely philosophical. This can break simulation-based proofs which require a simulator to commit to $\mathsf{F}(K, x)$, without knowing what x is, and then later provide a constrained key for x after learning x. It can also break proofs wherein an attacker can somehow require a CPRF reduction to act based on $\mathsf{F}(K, x)$ before the attacker has committed to whether x will be challenged or constrained.

We aim to bridge this gap with a definition that we call SIM*-AC-CPRF (simulation under adaptive compromise) security, which directly captures this philosophically appealing property. The adversary can arbitrarily interweaves requests for $\mathsf{F}(K, x)$ and K_R values with no restrictions. In the real world, it is honestly given these values. In the simulated world, the outputs of $\mathsf{F}(K, x)$ are sampled uniformly at random if a constrained key for x has not been given previously. A simulated is allowed to pick the constrained key (after being told the random outputs chosen for any previously queried $x \in R$) and picks the output of $\mathsf{F}(K, x)$ if a constrain key for x has been given previously. This definition is intuitively similar to the "SIM-AC" definitions of Jaeger and Tyagi (e.g., SIM-AC-PRF [23] and SIM*-AC-PRF [22]).

This definition would be impossible in the standard model by a simple counting argument (mirroring, e.g., proofs in [23,26]). However, it can be avoided in programmable ideal models (random oracle model, ideal cipher model, ...). The SIM-AC definitions of Jaeger and Tyagi allow the simulator to completely control the ideal primitive. We follow the SIM*-AC definition of Jaeger [22] in which ideal primitive to have an explicit programming interface that *both* the adversary and simulator can query. This modification improves the composability of ideal model definitions [13,22]. Most of our proofs require this composability.

We envision the use of ideal model native definitions as best used as a sort of hedging. Typically one can prove some form of non-adaptive security of a protocol using IND*-NC-CPRF security of the CPRF and

adaptive security using SIM*-AC-CPRF security (and partially adaptive security using IND*-AC-CPRF). If both analyses come to similar conclusions (perhaps with proofs that "intuitively" use the same "core"), one might infer that this reflects the "true" relationship between the security of the protocol and the CPRF. If the idealization in some way break, we can at least fall back on the non-adaptive assurances of security.

Applications. To exhibit the usefulness of our new definition, we apply it to the analysis of a searchable symmetric encryption (SSE) scheme by Bost, Minaud, and Ohrimenko [11] and a puncturable key wrapping (PKW) scheme by Backendal, Günther, and Paterson [2]. Both use constrained PRFs to achieve forward security notions.

The analysis of the SSE scheme runs into the sort of issue we described for simulation-based proofs where a simulator must provide outputs to the CPRF without knowing the inputs, then later provide constrained keys after being told the inputs. Bost, et al. resolved this by hashing the output of the CPRF with a random oracle. This hashed CPRF essentially being used to achieve SIM*-AC-CPRF security, so we prove security of the construction when the hashed CPRF is replaced with any SIM*-AC-CPRF secure CPRF.

Backendal, Günther, and Paterson [2] analyze their PKW scheme in a setting where nonces are not allowed to repeat, but also motivate that it may be useful in a setting with nonce reuse. They point out that in the latter setting, the proof would run into the sort of issue we described above where a reduction must provide $\mathsf{F}(K, x)$ without knowing whether x will be challenged or constrained. Backendal, et al. explicitly note that using ideal models might solve this issue and leave analyzing this as future work. We provide this analysis, using showing security of their scheme in the nonce-reuse setting by assuming SIM*-AC-CPRF security of the CPRF.

Constructions. We study the usefulness of SIM*-AC-CPRF security from the other direction by proving that natural constructions of CPRFs indeed achieve such security (under appropriate assumptions of the lower-level building blocks).

Hashed CPRF Constructions. We first show that our SSE proof indeed modularizes the analysis of Bost, et al. by proving SIM*-AC-CPRF security of $H(F(K, \cdot))$ when H is a random oracle and F is a non-adaptive secure CPRF. We can construct a simulator that honestly picks a key for K and honestly simulates all uses of F. When a constrain query is made, it reprograms H to make $H(F(K, x)) = z$ holds for any prior evaluation queries that returned z on input x (where x is in the constraint set). Security holds unless the adversary queries H on an unconstrained $F(K, x)$ or two different outputs of F on different inputs collide. A reduction to IND* security of the CPRF shows this is unlikely

GGM and AMAC (Cascade Constructions). The most commonly used CPRF (indeed the one proposed for use in the applications we consider) is based on the GGM pseudorandom function [19], so a natural question is whether it achieves SIM*-AC-CPRF security assuming appropriate idealization. Then, e.g., $H(GGM(K, \cdot))$ in the SSE scheme could instead be replaced with simply $GGM(K, \cdot)$ without the extra hash function. We will show that this is indeed the case, though rather than analyzing it directly, we instead capture it as a special case of the cascade construction of a PRF.

The cascade construction of Bellare, Canetti, and Krawczyk [6] iterates an underlying PRF. In particular, $\mathsf{CSC}(k_0, (x_0, \ldots, x_n))$ computes $k_{i+1} \leftarrow f(k_i, x_i)$ for $i = 0, \ldots, n$, then returns k_{n+1}.[1] Used naively, this is vulnerable to length-extension attacks because the output of CSC on input $\boldsymbol{x} = (x_0, \ldots, x_n)$ can be used to compute the output of CSC for all inputs that $\boldsymbol{x}$ is a prefix of. Bellare, et al. proved CSC is secure as long as its queries are "prefix free". The vulnerability naturally leads CSC to being usable as a constrained PRF because $\mathsf{CSC}(k, \boldsymbol{x})$ can act as a constrained key for inputs that $\boldsymbol{x}$ is a prefix of. This was observed in the initial works introducing constrained PRFs [9,12,25]. IND*-NC-CPRF security follows by a standard direct hybrid argument, while IND*-AC-CPRF security has proven more complex; several works have aimed to more precisely understand its IND*-AC-CPRF security [18,21,24]. Hofheinz, Kastner, and Klein [21] were finally able to establish that IND*-AC-CPRF security follows from the PRF security of f using a complex rewinding argument, though the concrete bounds are worse that those of the IND*-NC-CPRF proof.

[1] GGM is equivalent to the special case of this when $x_i \in \{0, 1\}$. The prior work on constrained PRFs specifically considered GGM, but to the best of our knowledge the results port over to CSC so we will discuss them as if they considered CSC instead.

We provide a proof that SIM*-AC-CPRF security of CSC follows from SIM*-AC-PRF security of f. The two proof provide analogous concrete security bounds to the IND*-NC-CPRF proof (from different assumptions) and seem to "intuitively" use the same "core" ideas — though the SIM*-AC proof is still noticeably more subtle. We view this as philosophical evidence that the challenge in writing IND*-AC-CPRF proofs and the worsened concrete security is somehow an artifact of our current proof techniques, rather than a reflection of the "true" relationship of the security between CSC and its underlying PRF.

Our proof first considers the generic composition of two CPRFs F and H, as a construction $\mathsf{CMP}[\mathsf{F},\mathsf{H}]$ which evaluates as $\mathsf{F}(\mathsf{H}(K,x'),x'')$ on input (x',x''). It allows direct constrains of K using H or indirect constrains that first evaluate $K' = \mathsf{H}(K,x')$ and then constrain K' using F. We show that SIM*-AC-CPRF security of H and F imply that of the composition. Then we observe that CSC can be viewed as a repeated application of CMP via $\mathsf{CSC}_1 = f$ and $\mathsf{CSC}_{i+1} = \mathsf{CMP}[f,\mathsf{CSC}_i]$. Thus we can use a hybrid argument to establish security of CSC.

Notably, a simplified version of the CMP proof provides the analogous result for IND*-NC-CPRF while it is not clear how to write such a proof for IND*-AC-CPRF security. Thus the simple CMP construction seems to contain at least some of the essential hardness of analyzing GGM's IND*-AC-CPRF security. The challenge arises when the adversary asks for the F key $K' = \mathsf{H}(K,x')$ to be constrained on a set R. The reductions don't know whether the adversary will later make a challenge query for some (x',x'') with $x'' \notin R$ (in which case K' needs to be switched to random) or request for H to constrain K on a set containing x' (in which case K' must not have been switched to random).

Next we consider AMAC. It is defined by $\mathsf{AMAC}(K,x) = \mathsf{Chop}(H(K||x))$ where H is a hash function and Chop is an unkeyed output transform such as truncation. We focus on the special case whose PRF security was analyzed by Bellare, Bernstein, and Tessaro [5] wherein H is a MD hash function with compression function $f(k,x)$ so $\mathsf{AMAC}(K,(x_0,\ldots,x_n))$ computes $k_1 \leftarrow f(x_0,K)$, then $k_{i+1} \leftarrow f(k_i,x_i)$ for $i = 1,\ldots,n$, then returns $\mathsf{Chop}(k_{n+1})$. I.e., it is basically the same as CSC except that the inputs for the first f call are flipped and Chop is applied to the output. Using Chop prevents length-extension attacks and maintains security even when inputs are prefixes of each other, but the k_i's can still be used as constrained keys. Bellare, et al. established security of AMAC as a PRF under appropriate assumptions on f.

We do the same for SIM*-AC-CPRF of AMAC from SIM*-AC-PRF security of f. In fact, we set up our formalism so that is essentially follows from the same analysis as CSC. We defined CMP so that on a "special" subset of H's domain it will merely output $\mathsf{H}(K,x')$ and not use F. (We prove security of CSC for arbitrarily prefix-free domains rather than a domain of all inputs of a fixed length by exploiting this feature of CMP as well.) Appropriate choice of this special subset captures AMAC as a repeated application of CMP.

1.2 Outline of the Paper

Section 2 provides general notation and specifies how we formalize programmable ideal models. Section 3 recalls existing security definitions for PRFs and CPRFs, then provides our new SIM*-AC-CPRF notion for CPRFs. Section 4 summarizes our analysis of searchable symmetric encryption and puncturable key wrapping using SIM*-AC-CPRF security. Section 5 proves the SIM*-AC-CPRF security of specific CPRF constructions.

2 Preliminaries

Sets, Tables, Lists. The empty set is $\emptyset$. The size of set S is $|S|$. A table T is written via $T[x] \leftarrow y$ and read via $y \leftarrow T[x]$. The empty table is $[\cdot]$. By $T[S]$ or T_S we denote a new table T' in which $T'[x] = T[x]$ if $x \in S$ and $T'[x] = \perp$ otherwise. The domain of T, denoted $T.D$ is the set of entries x at which $T[x] \neq \perp$.

If D is a set, we define $\mathsf{D}^0 = \{\varepsilon\}$, $\mathsf{D}^i = \mathsf{D}^{i-1} \times \mathsf{D}$, $\mathsf{D}^{\leq i} = \bigcup_{j \leq i} \mathsf{D}^j$, $\mathsf{D}^{<i} = \mathsf{D}^{\leq i-1}$, and $\mathsf{D}^* = \bigcup_{j \in \mathbb{N}} \mathsf{D}^i$. For $x,y \in \mathsf{D}^*$, the expression $x \sqsubseteq y$ denotes that x is a prefix of y. The empty list is $\emptyset$. If M is a list, then $M.\mathsf{add}(x_1,\ldots,x_n)$ appends the tuple $(x_1,\ldots,x_n)$ to the end of the list.

Pseudocode Conventions. We use a pseudocode formalism inspired by Bellare and Rogaway [8]. If A is a stateful, then $y \leftarrow A^O(x:\sigma)$ denotes assigning to y the output of A on input x and state σ with oracle access to O. The state is passed by reference, so A may internally update it. This notation is generalized to

multiple outputs, inputs, or oracles in a natural way. If A is randomized, then $\leftarrow\!\!\$$ is used in place of $\leftarrow$. Sets, tables, and lists are initialized empty.

We use an asymptotic formalism with security parameter λ which is passed to algorithms as the unary string 1^λ; it is sometimes left implicit for compactness of notation. The probability game G outputs true when run on λ is $\Pr[\mathsf{G}(\lambda)]$. Let $f : \mathbb{N} \to \mathbb{R}$. We say f is negligible if for all positive polynomials p there exists an $n \in \mathbb{N}$ such that $f(\lambda) \leq 1/p(\lambda)$ for all $\lambda \geq n$. We say f is super-polynomial if $1/f$ is negligible.

If S is a set, then $y \leftarrow\!\!\$\, S$ samples y uniformly from S. We often use a delayed sampling notation where $K_{(\cdot)} \leftarrow\!\!\$\, S$ means that each K_u is generated by $K_u \leftarrow\!\!\$\, S$ when first used.

We use $\perp$ to indicate some form of failure. "Require B" is shorthand for "If $\neg B$ then return $\perp$", i.e., B is required to be true to pass the require statement. Oracles that expect inputs from specific domains are implicitly assumed to immediately return $\perp$ if given invalid inputs. We sometimes use "Abort v" in the code for games and adversary. It means the current running process immediately halts and returns the corresponding value v.

Programmable Ideal Models. We work with Jaeger's [22] syntax for programmable ideal primitives (random oracles, ideal ciphers, random permutations, etc.). An ideal primitive P specifies initialization algorithm $\mathsf{P.Init}$, lazy sampling algorithms $\mathsf{P.Ls}$, and programming algorithm $\mathsf{P.Prog}$. Their syntaxes are $\sigma_\mathsf{P} \leftarrow\!\!\$\, \mathsf{P.Init}(1^\lambda)$, $y \leftarrow\!\!\$\, \mathsf{P.Ls}(1^\lambda, x : \sigma_\mathsf{P})$, and $\mathsf{P.Prog}(1^\lambda, x, y : \sigma_\mathsf{P})$.

Security games run $\mathsf{P.Init}$ at the beginning. Adversaries and simulators are allowed to query $\mathsf{P.Ls}(1^\lambda, x : \sigma_\mathsf{P})$ or $\mathsf{P.Prog}(1^\lambda, x, y : \sigma_\mathsf{P})$ for x, y of their choice. For compactness, we write this as a single oracle $\textsc{PPrim}(\mathsf{Op}, x, y)$ where $\mathsf{Op} \in \{\mathsf{Ls}, \mathsf{Prog}\}$ which runs $y \leftarrow\!\!\$\, \mathsf{P.Op}(1^\lambda, x, y : \sigma_\mathsf{P})$ and returns y. The extra input when $\mathsf{Op} = \mathsf{Ls}$ is ignored. Adversary access to programming can be important for simulation-based proofs as it allows the adversary to internally run simulators which expect programming access [13,22]. Cryptographic schemes may expect oracle access to $\mathsf{P.Ls}$; we write this as oracle access to P for compactness (in reductions, a schemes queries to P are be forwarded to the reductions $\textsc{PPrim}$ oracle if appropriate).

Jaeger [22] required that $\mathsf{P.Init}$ and $\mathsf{P.Ls}$ instantiate the lazy sampling of a function $f : \mathcal{D}_\lambda \to \mathcal{R}_\lambda$ according to a distribution $\mathcal{P}_\lambda$ where $\mathsf{P.Ls}(1^\lambda, x : \sigma_\mathsf{P})$ returns $f(x)$. Then programming function $\mathsf{P.Prog}(1^\lambda, x, y : \sigma_\mathsf{P})$ defines $f(x) = y$ as long as that is consistent with the definition of function up to that point. The random oracle model then is defined by P_rom where $\mathsf{P}_\mathsf{rom}.\mathsf{Init}$ outputs an empty table $\sigma_\mathsf{P} = [\cdot]$ and the other algorithms are as follows.

$\underline{\mathsf{P}_\mathsf{rom}.\mathsf{Ls}(1^\lambda, x : \sigma_\mathsf{P})}$	$\underline{\mathsf{P}_\mathsf{rom}.\mathsf{Prog}(1^\lambda, x, y : \sigma_\mathsf{P})}$
If $\sigma_\mathsf{P}[x] = \perp$ then $\sigma_\mathsf{P}[x] \leftarrow\!\!\$\, \mathcal{R}_\lambda$	If $\sigma_\mathsf{P}[x] = \perp$ then $\sigma_\mathsf{P}[x] \leftarrow y$
Return $\sigma_\mathsf{P}[x]$	Return $\perp$

Most of our proofs treat P as a black box and are agnostic to the particular primitive in use. Only for low-level analysis (e.g., PRF security of compression functions) will our analysis need to directly reason about P.

Cryptographic schemes may require multiple ideal primitives. Let P_0 and P_1 be ideal primitives. We define $\mathsf{P} = \mathsf{P}_0 \times \mathsf{P}_1$ via the following algorithms. In words, P runs parallel copies of each primitive. Each input includes a bit to specify which primitives should be used.

$\underline{\mathsf{P.Init}(1^\lambda)}$	$\underline{\mathsf{P.Ls}(1^\lambda, (d, x) : (\sigma_0, \sigma_1))}$	$\underline{\mathsf{P.Prog}(1^\lambda, (d, x), y : (\sigma_0, \sigma_1))}$
$\sigma_0 \leftarrow\!\!\$\, \mathsf{P}_0.\mathsf{Init}(1^\lambda)$	$y \leftarrow\!\!\$\, \mathsf{P}_\mathsf{d}.\mathsf{Ls}(1^\lambda, x : \sigma_d)$	$\mathsf{P}_\mathsf{d}.\mathsf{Prog}(1^\lambda, x, y : \sigma_d)$
$\sigma_1 \leftarrow\!\!\$\, \mathsf{P}_1.\mathsf{Init}(1^\lambda)$	Return y	Return $\perp$
Return (σ_0, σ_1)		

We say P is *consistent* if no combination of running $\mathsf{P.Ls}$ and $\mathsf{P.Prog}$ can make $\mathsf{P.Ls}(1^\lambda, x : \sigma_\mathsf{P})$ output $y' \neq y$ if it previously output y. Consistency is useful for knowing that deterministic algorithms which query $\mathsf{P.Ls}$ will be consistent in their own input-output behavior. The kind of lazy sampling primitives considered by Jaeger are consistent by definition. We say P is *non-programmable* if $\mathsf{P.Prog}$ does not update σ_P and the order in which $\mathsf{P.Ls}$ is run on different inputs does not change its output distributions. The ideal encryption model [32] is an example for which the latter condition fails; this failure can make proofs using it subtlely challenging [23].

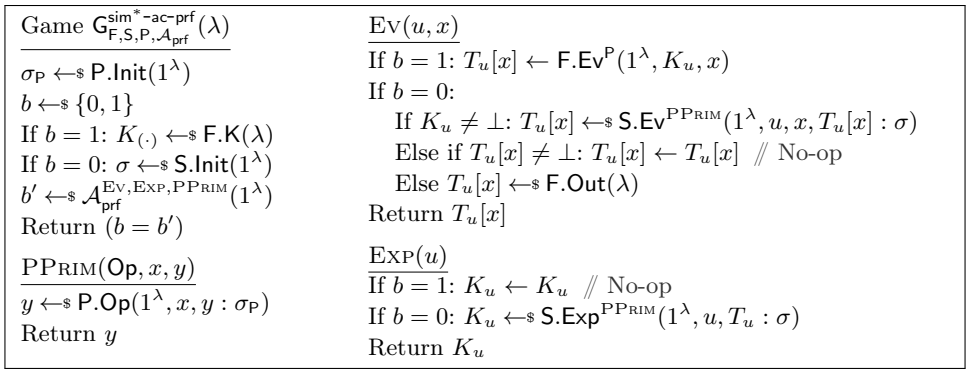


Fig. 1. Game defining the SIM*-AC-PRF security of F

3 (Constrained) PRFs: Old and New Definitions

In this section we recall existing definitions and introduce new definitions for pseudorandom functions (Section 3.1) and constrained pseudorandom functions (Section 3.2).

3.1 Function Families

A function family F specifies evaluation algorithm $\mathsf{F.Ev}$ together with sets $\mathsf{F.K}$, $\mathsf{F.Inp}$, and $\mathsf{F.Out}$. The syntax of its algorithm is $y \leftarrow \mathsf{F.Ev}^{\mathsf{P}}(1^\lambda, K, x)$ where $K \in \mathsf{F.K}(\lambda)$ is a key, $x \in \mathsf{F.Inp}(\lambda)$ is an input, and $y \in \mathsf{F.Out}(\lambda, x)$ is an output. We let $\mathsf{F.Fcs}$ denote the set of all possible functions f with domain $\mathsf{F.Inp}$ for which $f(x) \in \mathsf{F.Out}(x)$ always holds. We omit the x in $\mathsf{F.Out}(x)$ when the range is the same for all x. For compactness, we occasionally write $\mathsf{F}(K, x)$ in place of $\mathsf{F.Ev}^{\mathsf{P}}(1^\lambda, K, x)$.

SIM*-AC-PRF Security. SIM*-AC-PRF security [22] of F is captured by the game shown in Fig. 1. In the "real world" ($b = 1$), the attacker can ask for arbitrary PRF evaluations via $\textsc{Ev}$ and ask to expose keys via $\textsc{Exp}$. In the "simulated world" ($b = 0$), evaluation responses are random for unexposed users (because $K_u = \bot$ holds). On exposure, the simulator is given all input-output pairs from prior $\textsc{Ev}$ queries, then outputs a key. Its goal is to reprogram P to make the previous input-output pairs look consistent with F applied to this key. Intuitively, this definitions ensures PRF security holds until a user is exposed. We use the "no-op" operation $T_u[x] \leftarrow T_u[x]$ here (and similar "no-ops" elsewhere) just to emphasize that in this case we use the value which was already defined.

The advantage of $\mathcal{A}_{\mathsf{prf}}$ is $\mathsf{Adv}^{\mathsf{sim^*\text{-}ac\text{-}prf}}_{\mathsf{F},\mathsf{S},\mathsf{P},\mathcal{A}_{\mathsf{prf}}}(\lambda) = 2\Pr[\mathsf{G}^{\mathsf{sim^*\text{-}ac\text{-}prf}}_{\mathsf{F},\mathsf{S},\mathsf{P},\mathcal{A}_{\mathsf{prf}}}(\lambda)] - 1$. If there exists a PPT S such that $\mathsf{Adv}^{\mathsf{sim^*\text{-}ac\text{-}prf}}_{\mathsf{F},\mathsf{S},\mathsf{P},\mathcal{A}_{\mathsf{prf}}}$ is negligible for all PPT $\mathcal{A}_{\mathsf{prf}}$, then we say that F is SIM*-AC-PRF secure with P. Random oracles and ideal ciphers are known to be SIM*-AC-PRF secure [22,23].

By restricting $\mathcal{A}_{\mathsf{prf}}$ to never query $\textsc{Exp}$ we capture the typical notion of PRF security (in the programmable P model). We call this IND*-PRF security, use the superscript $\mathsf{ind^*\text{-}prf}$, and omit the simulator (which is never called).

This definition is tweaked from the SIM*-AC-PRF definition of [22] in that when $b = 1$ the latter will not re-run $\mathsf{F.Ev}^{\mathsf{P}}(1^\lambda, K_u, x)$ on repeated queries, instead it would output the stored value $T_u[x]$. This change is inspired by as of yet unpublished work. Therein we observe that the definitions are equivalent when P is consistent, but not when P is inconsistent. We argue that the definition we give here is more preferable for inconsistent P. Should not be considered to look like a random function if an attacker can cause its outputs to change (for unexposed keys).

SIM*-AC-PRF Security With Leakage. Bellare, Bernstein, and Tessaro [5] used a variant of PRF security for analyzing the augmented cascade and MAC function, wherein the attacker was given leakage $\mathsf{Chop}(K_u)$ for each user where $\mathsf{Chop} : \mathsf{F.K} \to \mathsf{Chop.Out}$. We will similarly need leakage for our SIM*-AC analysis of these constructions.

We model leakage by adding an oracle $\textsc{Leak}(u)$. This oracle will return $\mathsf{Chop}(K_u)$ whenever $K_u \neq \bot$ (meaning always in the real world and only if the user has been exposed in the simulated world). Otherwise in the simulated world it returns a random value (which is repeated on repeat queries). The simulator is told this random value if u is later exposed. The code is as follows, where $\ell_{(\cdot)} \leftarrow\!\!\$\ \mathsf{Chop.Out}(\lambda)$ is added to the beginning of the game when $b = 0$.

$\underline{\textsc{Leak}(u)}$	$\underline{\textsc{Exp}(u)}$
If $b = 1$: $L \leftarrow \mathsf{Chop}(1^\lambda, K_u)$	If $b = 1$: $K_u \leftarrow K_u$ // No-op
If $b = 0$: $L \leftarrow \ell_u$	If $b = 0$: $K_u \leftarrow\!\!\$\ \mathsf{S.Exp}^{\mathrm{PPRIM}}(1^\lambda, u, T_u, \ell_u : \sigma)$
Return L	Return K_u

Note that security with respect to this definition is equivalent to requiring SIM*-AC-PRF security of the function family $\mathsf{F}_{\mathsf{Chop}}$ which is identical to F except that we define $\mathsf{F}_{\mathsf{Chop}}.\mathsf{Ev}^{\mathsf{P}}(1^\lambda, K, \varepsilon) = \mathsf{Chop}(1^\lambda, K)$ where $\varepsilon \notin \mathsf{F.Inp}$. For simplicity, we will directly use security of $\mathsf{F}_{\mathsf{Chop}}$ in our results, rather than introducing explicit advantage function and security definition for SIM*-AC-PRF with leakage.

In Appendix G, we port proofs of Bellare, Bernstein, and Tessaro [5] over to the SIM*-AC setting to show that (under light combinatorial assumptions about Chop) a random oracle achieves security with key leakage, as does the Davies-Meyer construction based on an ideal cipher. Along the way, we identify and fix a bug in their Davies-Meyer proof. Our concrete security bounds mirror those of [5], achieving better than trivial multi-user security which aids the security bounds we will prove for AMAC as a constrained PRF.

3.2 Constrained Function Families

Constrained PRFs were introduced independently by Boneh and Waters [9]; Kiayias, Papadopoulos, Triandopoulos, and Zacharias [25] (termed "delegatable PRFs"); and Boyle, Goldwasser, and Ivan [12] (termed "functional PRFs"). Our syntax is closest to Boneh and Waters's definition. Constrained PRFs can create special "constrained" keys which allow evaluating the PRF on a restricted set of inputs while revealing nothing else.

Syntax. A constrained function family CF is a function family (with the syntax that follows) which additionally specifies a constrain algorithm $\mathsf{CF.Con}$, a constrained evaluation algorithm $\mathsf{CF.CEv}$, and a set system $\mathsf{CF.SS} \subseteq \mathcal{P}(\mathsf{CF.Inp})$. Here $\mathcal{P}$ denotes the power set. The syntax of these algorithms is $K_R \leftarrow \mathsf{CF.Con}^{\mathsf{P}}(1^\lambda, k, R)$ and $y \leftarrow \mathsf{CF.CEv}^{\mathsf{P}}(1^\lambda, K_R, R, \boldsymbol{x})$.

Here $k \in \mathsf{CF.K}(\lambda)$ is the root key, $R \in \mathsf{CF.SS}(\lambda)$ is a set, K_R is a constrained key for R, $\boldsymbol{x} \in R \subseteq \mathsf{CF.Inp}(\lambda)$ is an input, and $y \in \mathsf{CF.Out}(\lambda)$ is an output. The desired behavior is that $\mathsf{CF.Ev}^{\mathsf{P}}(1^\lambda, k, \boldsymbol{x}) = \mathsf{CF.CEv}^{\mathsf{P}}(1^\lambda, K_R, R, \boldsymbol{x})$ when $K_R \leftarrow \mathsf{CF.Con}^{\mathsf{P}}(1^\lambda, k, R)$ for all $\boldsymbol{x} \in R \in \mathsf{CF.SS}(\lambda)$. For technical reasons, we also need a query consistency property that the exact same set of oracle queries would be made on either side of this equality.

We find including R as an explicit input to constrained evaluation to be notationally convenient so that we do not have to encode it in K_R. If we cared about whether K_R hides the set R ("policy hiding" [25]), then this would be inappropriate. We assume that R is encoded as a bit string in a reasonable manner, e.g., checking $x \in R$ should be efficient given R.

When useful, we assume $\mathsf{CF.SS}$ contains the entire domain ($K_{\mathsf{CF.Inp}} = k$) and all singleton sets ($K_{\{\boldsymbol{x}\}} = \mathsf{CF.Ev}^{\mathsf{P}}(1^\lambda, k, \boldsymbol{x})$). In security games this allows us to omit explicit oracles for for revealing k or for non-challenge evaluation queries [9,25]. (Our Theorem 2 makes use of the latter.) One can also modify CF so that $\mathsf{CF.SS}$ is closed under unions ($K_{R_1 \cup R_2} = (K_{R_1}, K_{R_2})$).

We say that CF is a prefix-contrained function family if $\mathsf{CF.Inp}$ is a subset of D^* for some set D and $\mathsf{CF.SS}$ contains all sets $R_{x\sqsubseteq} = \{y \in \mathsf{CF.Inp} \mid x \sqsubseteq y\}$ for $x \in \mathsf{D}^*$. Recall $x \sqsubseteq y$ denotes that x is a prefix of y. Natural choices are $\mathsf{CF.Inp} = \mathsf{D}^*$, $\mathsf{CF.Inp} = \mathsf{D}^n$, or $\mathsf{CF.Inp} = \mathsf{D}^{\leq n}$ for some integer n. It is well known [9,12,21,25,18] that the classic Goldreich-Goldwasser-Micali (GGM) construction of a PRF [19] naturally induces a prefix-contrained function family.

Reconstrainability. A constrained function family CF is reconstrainable if it additionally specifies a constrained constrain algorithm $\mathsf{CF.CCon}$ for placing further constraints on an already constrained key. Its syntax is $K_{R'} \leftarrow \mathsf{CF.CCon}^{\mathsf{P}}(1^\lambda, K_R, R, R')$. Here $R' \subset R$ are sets in $\mathsf{CF.SS}(\lambda)$, K_R is a constrained key for R, and $K_{R'}$ is a constrained key for R'.

A useful property of a reconstrainable constrained function family is constrain invariance which, roughly, requires that a constrained key for R' is the same whether it is output by $\mathsf{CF.Con}$ or $\mathsf{CF.CCon}$.

Constrained PRF Security. The standard notion of constrained PRF security for a constrained function family CF (with a programmable ideal primitive) is shown by the game $\mathsf{G}^{\mathsf{ind^*\text{-}ac\text{-}cprf}}$ defined in Fig. 2. The game captures that outputs of CF should look random, unless the attacker has learned a constrained key from which that output could be evaluated. We have written it in the standard single-bit, multi-user style.

In the game, the constrain oracle Con uses $\mathsf{CF.Con}$ to compute constrained keys. The evaluation (challenge) oracle Ev uses $\mathsf{CF.Ev}$ in the "real world" ($b = 1$) and the output of truly random functions f_u in the "random world" ($b = 0$). The attacker's goal is to distinguish which of these two cases is happening. To prevent trivial attacks, we use sets $C_u \subseteq \mathsf{CF.Inp}$ and $E_u \subseteq \mathsf{CF.Inp}$ for each user. The former tracks all points for which the adversary can compute $\mathsf{CF.Ev}$ via the constrained keys it has received. The latter tracks all points for which the adversary has received a challenge. Trivial attacks are prevented by maintaining the invariant that $C_u \cap E_u = \emptyset$.

The advantage of $\mathcal{A}_{\mathsf{cprf}}$ is $\mathsf{Adv}^{\mathsf{ind^*\text{-}ac\text{-}cprf}}_{\mathsf{CF},\mathsf{P},\mathcal{A}_{\mathsf{cprf}}}(\lambda) = 2\Pr[\mathsf{G}^{\mathsf{ind^*\text{-}ac\text{-}cprf}}_{\mathsf{CF},\mathsf{P},\mathcal{A}_{\mathsf{cprf}}}(\lambda)] - 1$. If $\mathsf{Adv}^{\mathsf{ind^*\text{-}ac\text{-}cprf}}_{\mathsf{CF},\mathsf{P},\mathcal{A}_{\mathsf{cprf}}}$ is negligible for all PPT $\mathcal{A}_{\mathsf{cprf}}$, then we say that PF is IND*-AC-CPRF secure with P.

The above represents adaptive security, as the adversary can arbitrarily interweave constrain and evaluation queries. Non-adaptive security would require evaluation queries be made before constrain queries. This is captured by adding the boxed code to the game.[2] We call this IND*-NC-CPRF security and use the superscript $\mathsf{ind^*\text{-}nc\text{-}cprf}$.

SIM*-AC Constrained PRF Security. While IND*-AC-CPRF security is an adaptive security notion, it still embeds some inherent non-adaptivity. If an attacker needs to know $y = \mathsf{CF.Ev}(1^\lambda, k_u, \boldsymbol{x})$, then it must commit to whether this will be a challenge point (by querying $\textsc{Ev}(u, \boldsymbol{x})$) or a non-challenge point that may be part of later constrain sets (by querying $\textsc{Con}(u, \{\boldsymbol{x}\})$). This does not (conveniently) allow capturing the natural property that each y should look random up until a point of time when it is exposed by a constrain key. To capture this intuitive property, we cast constrained PRF security into the SIM*-AC setting. Consider the security game $\mathsf{G}^{\mathsf{sim^*\text{-}ac\text{-}cprf}}$ shown in Fig. 2 for constrained function family CF. In this code, for a set S we let $f_u(S)$ denote some representation (e.g., as a table) of the restricted function $f' : S \to \mathsf{CF.Out}$ defined by $f'(x) = f_u(x)$.

The "real world" of this security game ($b = 1$) is identical to that of IND*-AC-CPRF except that no restrictions are place on what queries the adversary can make. In the "simulated world" ($b = 0$) the outputs of both oracles are simulated, either being defined by the truly random functions f_u or chosen by a simulator S. In particular, whenever the evaluation on $\boldsymbol{x}$ should not trivially be known by the adversary (i.e., when $\boldsymbol{x} \notin C_u$) it is chosen at random. Otherwise the simulator is told the query, so it can simulate the response appropriately. The simulator always controls the output of constrain queries. To prevent trivial attacks, we reveal to the simulator the previously random outputs of evaluation on points which are contained in this constrain query. Intuitively, the definition requires that each $y = \mathsf{CF.Ev}(1^\lambda, k_u, \boldsymbol{x})$ looks random until a relevant constrain query is made, at which point the simulator makes it look like y was derived from the constrain key it chooses.

The advantage of $\mathcal{A}_{\mathsf{cprf}^*}$ is $\mathsf{Adv}^{\mathsf{sim^*\text{-}ac\text{-}cprf}}_{\mathsf{CF},\mathsf{S},\mathsf{P},\mathcal{A}_{\mathsf{cprf}^*}}(\lambda) = 2\Pr[\mathsf{G}^{\mathsf{sim^*\text{-}ac\text{-}cprf}}_{\mathsf{CF},\mathsf{S},\mathsf{P},\mathcal{A}_{\mathsf{cprf}^*}}(\lambda)] - 1$. If there exists a PPT S such that $\mathsf{Adv}^{\mathsf{sim^*\text{-}ac\text{-}cprf}}_{\mathsf{CF},\mathsf{S},\mathsf{P},\mathcal{A}_{\mathsf{cprf}^*}}$ is negligible for all PPT $\mathcal{A}_{\mathsf{cprf}^*}$, then we say that CF is SIM*-AC-CPRF secure with P.

In Section 4, we analyze several applications of constrained function families and show they require full SIM*-AC-CPRF security. In Section 5, we give examples of constrained function families that achieve SIM*-AC-CPRF security (under appropriate assumptions on their building blocks).

[2] Note that this also allows evaluation queries after constrain queries if the evaluation query was already made previously. For consistent P the later query must anyway match the earlier one and for non-programmable P the later query could simply have been made earlier. For other P, the ability to make more queries potentially provides the adversary with extra powers.

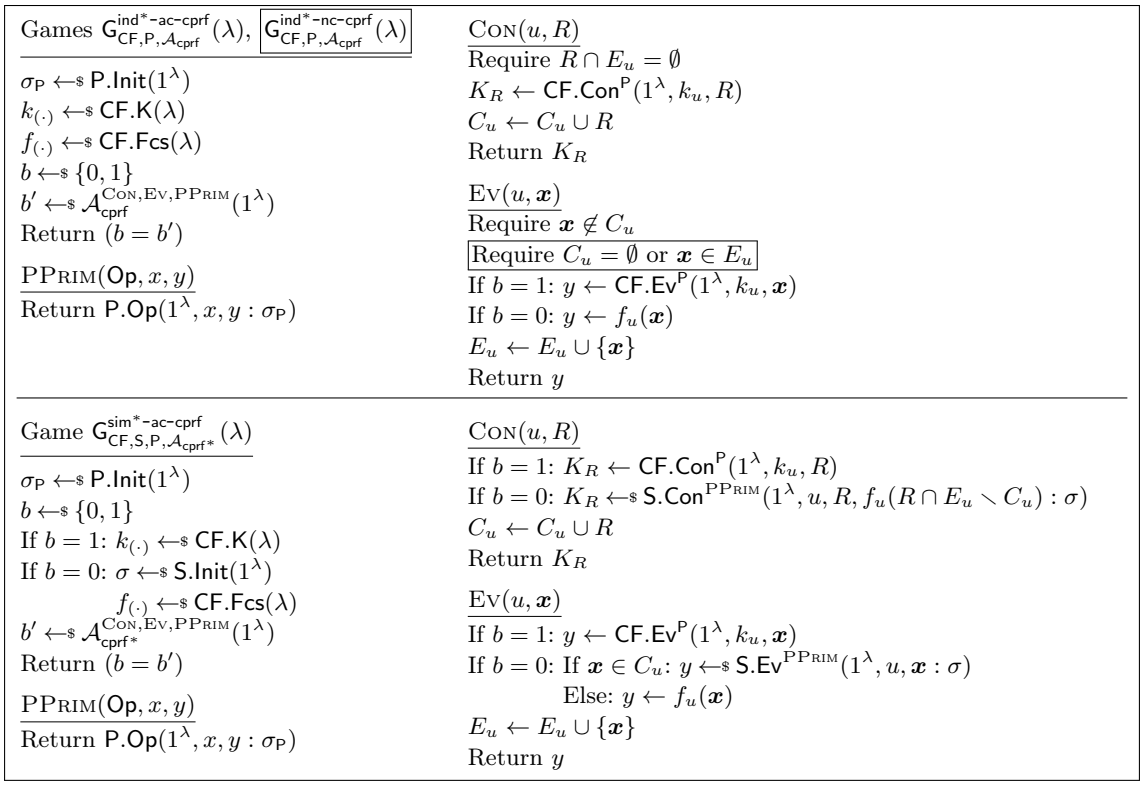


Fig. 2. Games defining constrained PRF security and SIM*-AC-CPRF security of CF. Boxed code is only included for non-adaptive security. If S is a set, $f_u(S)$ is the restriction of function f_u to domain S.

3.3 Simple Security Claims

SIM*-AC implies IND*-AC. We have described SIM*-AC-CPRF security as if it is stronger than IND*-AC-CPRF security. The following theorem formalized this. In the other direction, IND*-AC-CPRF security will not imply SIM*-AC-CPRF security security as the former is achievable in the standard model [9,12,25] while a natural counting argument will show that the latter is not [23,26].

Theorem 1. *If* CF *is SIM*-AC-CPRF secure with* P*, then it is IND*-AC-CPRF secure with* P*. For all* $\mathcal{A}_{\mathsf{cprf}}$*, we can build* $\mathcal{A}_{\mathsf{cprf}^*}$ *such that for all simulators* S

$$\mathsf{Adv}^{\mathsf{ind^*\text{-}ac\text{-}cprf}}_{\mathsf{CF},\mathsf{P},\mathcal{A}_{\mathsf{cprf}}}(\lambda) = 2 \cdot \mathsf{Adv}^{\mathsf{sim^*\text{-}ac\text{-}cprf}}_{\mathsf{CF},\mathsf{S},\mathsf{P},\mathcal{A}_{\mathsf{cprf}^*}}(\lambda).$$

The main idea is that if $\mathcal{A}_{\mathsf{cprf}}$'s queries are forwarded to $\mathcal{A}_{\mathsf{cprf}^*}$'s oracles, then the restrictions placed on $\mathcal{A}_{\mathsf{cprf}}$'s queries will mean that the else branch in $\textsc{Con}$ for SIM*-AC-CPRF will never be triggered so y is always $f_u(\boldsymbol{x})$, which will not be used elsewhere. The proof is delayed to Appendix A.

CPRFs from PRFs. Let F be a function family. Then we will sometimes view F as a simple constrained function family whose set system is only the entire domain ($K_{\mathsf{CF.Inp}} = k$) and all singleton sets ($K_{\{\boldsymbol{x}\}} = \mathsf{CF.Ev}^{\mathsf{P}}(1^\lambda, k, \boldsymbol{x})$), with the natural constrained evaluation algorithm. The proof of the following is straighforward.

Proposition 1. *For* $X \in \{$*IND*-NC, IND*-AC, SIM*-AC*$\}$*,* F *is X-CPRF secure with* P *if and only if it is X-PRF secure with* P*.*

4 Applications of SIM*-AC-CPRF Security

We exhibit the usefulness of our SIM*-AC-CPRF definition by describing its use in two applications.

4.1 Searchable Symmetric Encryption

Searchable symmetric encryption (SSE) aims to store databases of documents in an encrypted form that nonetheless allows search queries to be performed to identify all documents that are associated with a given keyword. The most efficient protocols aim to have search only take a single round trip. The user sends a search token τ_s for keyword w to the server who then uses an encrypted index to recover all document indices ind associated with w. The server can then send the document ciphertexts associated with each index back to the user who decrypts the ciphertexts to read their document. Such one-round protocols lead to adaptive security challenges in the standard simulation-based definition introduced by Curtmola et al. [16].

A Common Design Pattern. A common design paradigm for SSE schemes involves associating an encryption key ek_w and a PRF key k_w with each keyword. The encryption key encrypts the document identifiers and the PRF key determines random-looking places that these ciphertexts will be stored.

A clean example of this is the Π_{bas} scheme of Cash et al. [14] where (roughly) the i-th document identifier id associated with w is stored in a dictionary via $T[\mathsf{F}(k_w, i)] \leftarrow \mathsf{Enc}(ek_w, id)$. Given ek_w and k_w, the server loops over i, recovering each ciphertext from T and decryption. Cash et al. [14] proved non-adaptive security of this construction assuming IND-PRF security and that the encryption provides indistinguishability from random strings and sketched an adaptive proof based on random oracles.

Jaeger and Tyagi [23] identified a number of SSE schemes that used similar techniques and showed that SIM-AC-PRF security of the function family and SIM-AC-KP (a "key private" variant of CPA security in the SIM-AC style) security of encryption would have sufficed in the analysis, allowing simpler and more modular proofs. As an example, they gave a proof for Π_{bas}.

Adding Forward Security. More advanced constructions of SSE aim for strong security properties like forward security. Searching for keyword w at time t should not allow the server to use the search token to identify documents associated with the keyword that are added to the database later. Constrained PRFs have shown themselves to be a natural tool for achieving this and other properties for SSE [10,11,17,25,30,31,33,34].

Perhaps the canonical example of constructing a forward-secure SSE scheme from constrained PRFs is the FS-RCPRF scheme of Bost, Minaud, and Ohrimenko [11] which we show in Fig. 3 (with some details omitted or simplified). The construction is based on two hash functions H_0 and H_1 and a CPRF $\tilde{\mathsf{F}}$ with $\tilde{\mathsf{F}}.\mathsf{Inp} = \mathbb{N}$ which can constrain its key on sets $[c] = \{0, ..., c\}$.

The server stores a table EDB mapping from pseudorandom labels UT to ciphertexts e which contain the document identifier. The client stores keys K_w and K'_w for each w and a table $\mathbf{W}$ with a counter of how many documents associated with w are already stored in the database. The update protocol allows the client to add a new document identifier ind to keyword w. It applies $\tilde{\mathsf{F}}$ with K_w to the next counter value to obtain an intermediate value T which is hashed together with K'_w to derive UT and encrypt ind into e. The update token $\tau_u = (UT, e)$ is then stored by the server in EDB.

To perform a search for keyword w, the client constrains K_w to only be valid on the counter values that have been used for w so far. This, together with K'_w allows the server to recompute the UT values, then recover and decrypt the ciphertexts e.

A natural question is why this construction needs the hash functions at all. Shouldn't the output of $\tilde{\mathsf{F}}$ already look random for our purposes? The answer is that the hashes (modelled as random oracles) are incorporated to allow an adaptive security proof where the simulator commits to a random EDB and then later reprograms these random oracles to make it look consistent with the attacker's queries.

We view the extra hashes as, in essence, a way of lifting from non-adaptive to adaptive (in particular SIM*-AC-CPRF) security. However, with this perspective it is natural to omit the hashes and view the construction in terms of a single SIM*-AC-CPRF secure function. We capture this in the construction $\mathsf{FS2}$ shown on the bottom of the figure. We additionally removed K'_w entirely (it does not seem to add anything and was unused in the security analysis of $\mathsf{FS\text{-}RCPRF}$) and replace one-time pad encryption with arbitrary

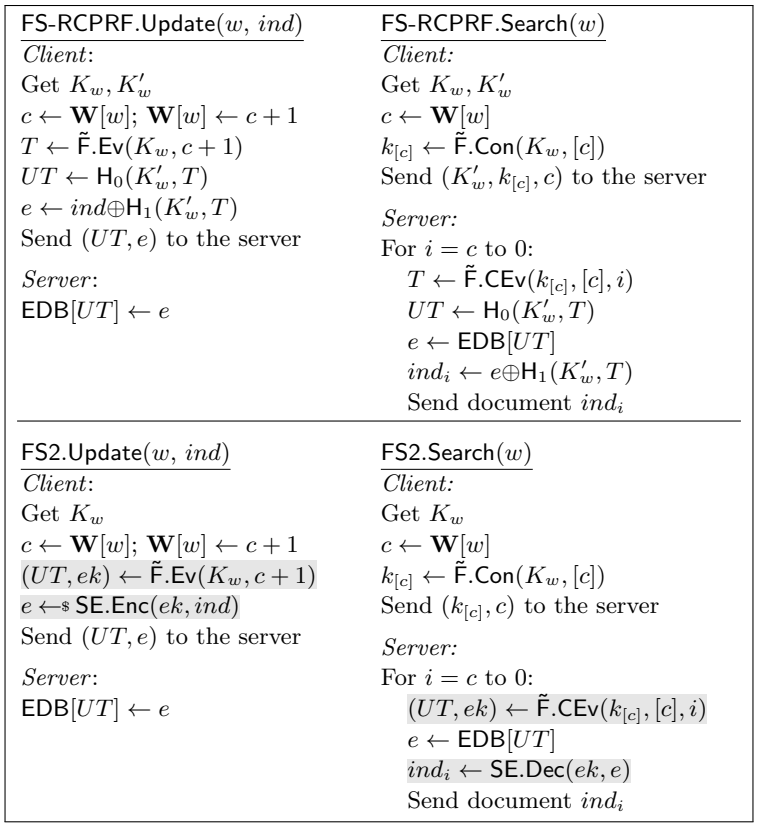


Fig. 3. SSE scheme FS-RCPRF and our simplification/modularization FS2. Differences to FS-RCPRF are highlighted.

one-time secure encryption scheme. This allows, e.g., swapping in standard encryption schemes that provide authentication.

In Appendix B, we formalize this construction and prove that it is secure assuming $\tilde{\mathsf{F}}$ is SIM*-AC-CPRF security and SE is SIM*-AC-CPA secure (a CPA security notion in the SIM*-AC style). This has several benefits. The proof is easier than the security proof for FS-RCPRF, in particular avoiding the need to handle random oracle reprogramming analysis in the middle of an already complicated proof. Indeed, there is a minor bug in this step of the Bost, et al. [11] proof. (They did not consider the possibility that two different honestly generated T's could collide.)

The result provides modularity; we have more precisely identified exactly what properties are required of the subcomponents so that implementers have more flexibility. To capture the original scheme (without K'_w), in Section 5 we formalize and analyze hashing the output of a non-adaptively secure CPRF to obtain an adaptively secure one. This works as long as the adversary's queries are restricted to an a priori known small (polynomial sized) domain. In FS-RCPRF, it is bounded by the number of update operations performed. The primary instantiation proposed by Bost, et al., called Diana, uses $\tilde{F}$ based on the Goldreich, Goldwasser, Micali (GGM) CPRF [9,12,19,25]. Our separate analysis of this construction in Section 5 (as a special case of the cascade construction) shows that it achieves SIM*-AC-CPRF security and the extra hashes in Diana were unnecessary.

We believe that our definition would have been applicable to a number of other SSE constructions that use similar techniques [17,30,33,34].

4.2 Puncturable Key Wrapping

Next we consider puncturable key wrapping. Backendal, Günther, and Paterson [2] introduced puncturable key wrapping (PKW) as a means to add forward security to key wrapping and then used this to analyze a

variant of the session resumption mechanism in the pre-shared key mode of TLS 1.3 introduced by Aviram, Gellert, and Jager [1] and an encrypted file storage system that they specify.

A puncturable key wrapping scheme PKW provides a key generation algorithm PKW.Kg, wrapping (i.e., encryption) algorithm PKW.Wrap, unwrapping (decryption) algorithm PKW.Unwrap, and puncturing algorithm PKW.Punc. The syntaxes of these algorithms are $K \leftarrow \mathsf{PKW.Kg}(1^\lambda)$, $c \leftarrow \mathsf{PKW.Wrap}^{\mathsf{P}}(1^\lambda, K, t, h, m)$, $m \leftarrow \mathsf{PKW.Unwrap}^{\mathsf{P}}(1^\lambda, K, t, h, c)$, and $K' \leftarrow \mathsf{PKW.Punc}^{\mathsf{P}}(K, t)$. Here t is a tag (nonce), h is a header to be authenticated (associated data), and m is a message (which is expected to be a key, though the formalism works equally well for the encryption of arbitrary data) to be wrapped. The unwrapping algorithm recovers the message from the ciphertext. The puncturing algorithm creates an updated key which is no longer able to encrypt or decrypt with tag t and can be applied multiple times.

Backendal, et al. introduced several integrity and confidentiality definition of PKW schemes. We focus on their strongest confidentiality definition FIND$-RCPA which allowed the attacker to ask for real wrappings, real or random challenge wrapping, and punctured keys of their chosen with the restriction that they can never repeat tags against wrapping queries or ask for keys which have not been punctured on all of the tags that will be used for challenge wrappings. They propose a natural construction $\mathsf{PKW}[\mathsf{CF}, \mathsf{NE}]$ which applies a (reconstrainable) constrained PRF to the tag to derive a subkey which is then used with an AEAD scheme to perform the actual encryption. They then prove that $\mathsf{PKW}[\mathsf{CF}, \mathsf{NE}]$ is FIND$-RCPA secure assume CF is IND-AC-CPRF secure and NE achieves the standard IND$-CPA security notion for nonce-based encryption [27].[3]

However, Backendal, et al. also note that their formalism does not allow the use of "batch puncturing" [20] (removing the ability to decrypt a collection of ciphertexts with a single puncture operation) because FIND$-RCPA provides no assurances if the same tag is used for multiple ciphertexts. Backendal et al. propose but do not formally analyze a stronger notion where tag reuse is allowed [2, Footnote 2, ePrint]. They point out that misuse-resistant security of NE [28] would be a necessary condition for $\mathsf{PKW}[\mathsf{CF}, \mathsf{NE}]$ to be secure in this model. Moreover, they observe that the natural proof strategy would run into a commitment problem. When an adversary asks for a real encryption of (t, h, m), we want to provide a real encryption if a key capable of decrypting nonce t will later be revealed or switch the encryption to random if t will later be used for a challenge encryption (preventing such a key being revealed). Backendal et al. proposed that the stronger notion may be achieved by using ideal model analysis (citing the SIM-AC paper [23]), but leave doing so for future work. We confirm this belief. Interestingly, in our analysis we require SIM*-AC-based idealization of *both* CF and NE. We give the formal details in Appendix C.

5 Adaptively Secure Constructions

In this section, we analyze the security of several constrained function families with respect to our new SIM*-AC-CPRF definition. In Section 5.1, we analyze the construction that is used in FS-RCPRF from Bost, et al. [11]. Then in Section 5.2, we analyze the construction obtained from composing two constrained PRF together. This allows us to analyze cascade construction in Section 5.3. Finally, we analyze AMAC in Section 5.4, which can be viewed as an instantiation of augmented cascade that we considered in Section 5.3.

5.1 Hashed Constrained PRF(s)

We start with the construction we view as being implicit in the searchable encryption scheme FS-RCPRF from Bost, et al. [11] which lifts non-adaptive constrained PRF security to be adaptive. Related constructions are also discussed in [30]. In essence, if CF is a constrained function family with key K and H is a hash function modeled as a random oracle with key K', their construction computes outputs via $H(K', \mathsf{CF}(K, \cdot))$.

The second K' does not seem to provide much additional value (e.g., the security proof of FS-RCPRF [11] does not make use of this key). Thus we will simplify and obtain the construction $\mathsf{Hashed}[\mathsf{CF}]$ defined identically to CF except that CF.Out is always the range of random oracle $\mathsf{P}_{\mathsf{rom}}$ and evaluation/constrained

[3] Technically, they formalize this in terms of a puncturable PRF. This is essentially just a difference of syntax, though the security notion they use for it is technically slightly weaker than IND*-AC-CPRF security because it does not allow the attacker to access as many constrained keys.

$\underline{\mathsf{S.Init}(1^\lambda)}$
$k_{(\cdot)} \leftarrow\!\!\$\ \mathsf{CF.K}(\lambda)$
Return $k_{(\cdot)}$

$\underline{\mathsf{S.Ev}^{\mathrm{PPRIM}}(1^\lambda, u, x : k)}$
$y \leftarrow \mathsf{Hashed[CF].Ev}^{\mathsf{P}\times\mathsf{P}_{\mathsf{rom}}}(1^\lambda, k_u, x)$
Return y

$\underline{\mathsf{S.Con}^{\mathrm{PPRIM}}(1^\lambda, u, R, f : k)}$
For $x \in f.\mathsf{D}$:
  $Y \leftarrow \mathsf{CF.Ev}^{\mathsf{P}}(1^\lambda, k_u, x)$
  $\mathrm{PPRIM}(\mathsf{Prog}, (1, Y), f(x))$
Return $\mathsf{CF.Con}^{\mathsf{P}}(1^\lambda, k_u, R)$

Fig. 4. Simulator S for the SIM*-AC-CPRF security proof of $\mathsf{Hashed[CF]}$ (Theorem 2)

evaluation both apply $\mathsf{P}_{\mathsf{rom}}$'s to their output.

$$\mathsf{Hashed[CF].Ev}^{\mathsf{P}\times\mathsf{P}_{\mathsf{rom}}}(1^\lambda, k, \boldsymbol{x}) = \mathsf{P}_{\mathsf{rom}}.\mathsf{Ls}(1^\lambda, \mathsf{CF.Ev}^{\mathsf{P}}(1^\lambda, k, \boldsymbol{x}) : \sigma_{\mathsf{P}})$$
$$\mathsf{Hashed[CF].CEv}^{\mathsf{P}\times\mathsf{P}_{\mathsf{rom}}}(1^\lambda, K_R, R, \boldsymbol{x}) = \mathsf{P}_{\mathsf{rom}}.\mathsf{Ls}(1^\lambda, \mathsf{CF.CEv}^{\mathsf{P}}(1^\lambda, K_R, R, \boldsymbol{x}) : \sigma_{\mathsf{P}})$$

Security of this construction boils down to arguing that the attacker cannot query $\mathsf{P}_{\mathsf{rom}}$ (for lazy sampling or programming) on the outputs of CF and that outputs of CF do not collide before the attacker trivially learned the outputs via constrain queries. This should be hard because these output of CF should look random. To prove security of this construction based on non-adaptive security of CF, we will need to guess ahead of time the input to CF whose output the attacker will query to $\mathsf{P}_{\mathsf{rom}}$ and so security will require restricting attention to attacks that only query for evaluation from sets of polynomial size. In the case of FS-RCPRF [11] each attacker would only learn CF outputs for inputs from a set $\{1, \ldots, c\}$ where c corresponded to the maximum number of documents associated with a given keyword in an attacker chosen database. This value could depend on the attacker, but was necessarily polynomial. The following theorem captures the security of this construction.

Theorem 2. *If* $\min_x |\mathsf{CF.Out}(\lambda, x)|$ *is super-polynomial,* CF *is IND*-NC-CPRF secure with* P*, and* P *is non-programmable; then* $\mathsf{Hashed[CF]}$ *is SIM*-AC-CPRF secure with* $\mathsf{P} \times \mathsf{P}_{\mathsf{rom}}$ *against adversaries whose* Ev *queries are restricted to a set of polynomial size.*

The simulator we construct for proving this theorem is given in Fig. 4. It simulates the use of CF honestly and in $\mathsf{S.Con}$ attempts to reprogram the random oracle to make it consistent which the values returned by earlier evaluation queries. The simulation will fail if $\mathsf{P}_{\mathsf{rom}}$ is queried on $y = \mathsf{CF.Ev}^{\mathsf{P}}(1^\lambda, k_u, \boldsymbol{x})$ for an unconstrained x. This can be caused by a direct lazy sampling or programming query by the attacker or if a unrelated evaluation query happened to compute the same y. (The latter condition was missed in security proof for FS-RCPRF [11].) To bound this via a non-adaptive reduction, which knows the set of queries N that the adversary may make, will pick a random $(u, \boldsymbol{x}) \leftarrow\!\!\$\ N(\lambda)$ then query $\mathrm{Ev}(u, \boldsymbol{x})$ at the start of the game to get $\boldsymbol{y}$, and then simulate the rest of the game (aborting early if (u, x) will be constrained). It returns 1 if $\boldsymbol{y}$ is queried to $\mathsf{P}_{\mathsf{rom}}$ (directly or via a different Ev query). The technical condition that P be non-programmable arises from the fact that the reduction will obtain $\boldsymbol{y}$ at the start of the game (so that it is non-adaptive) even though it may not actually be required until a later query. If P were programmable, then it's possible that queries to P before the later query should have changed the distribution of $\boldsymbol{y}$. We defer the proof to Appendix D.1. This theorem does not allow CF to depend on the $\mathsf{P}_{\mathsf{rom}}$ used by $\mathsf{Hashed[CF]}$.

Instantiations. The searchable encryption schemes Diana [11] and *Σοφοζ* [10] can be viewed as essentially using instantiations of hashed CPRF. Diana uses GGM as the underlying constrained PRF, therefore it can be analyzed as a $\mathsf{Hashed[CF]}$ and security follows from the known non-adaptive constrained PRF security of GGM [9,12,21,25,18]. However, in Section 5.3 we will show that the cascade construction (a generalization of GGM) is already SIM*-AC-CPRF secure, assuming SIM*-AC-PRF security of the underlying building block. So GGM could have been used directly, without hashing. Moreover, relying on that result will provide stronger concrete security guarantees because it avoids the guessing argument.

Σοφοζ is analyzed as an instantiation of FS-RCPRF by Bost, et al. [11] by constructing constrained PRF using iterations of a trapdoor permutation (TDP). The proof strategy claimed by Bost, et al. is flawed, as the constructed CPRF is not secure. However, the overall hashed construction is nonetheless secure as we will now show.

Trapdoor permutation constrained "PRF". We consider the construction in [11]. Let Π be a trapdoor permutation permutation with trapdoor tk and public information pub. For all valid input ST, $\Pi_{pub}(ST)$ computes the permutation using pub, and $\Pi^{-1}_{tk}(ST)$ computes the inverse using tk. Let Π^n or Π^{-n} denote the n-fold iteration of these algorithms. (More formal definitions will be given for the coming security result.) Then the constrained function family $\tilde{\mathsf{F}}$ is as follows. Its key k is composed of tk and some initial secret input ST_0. The constrain sets are $R_n = \{1^0, \ldots, 1^n\} = \{1^i : i \leq n\}$. Its algorithms are:

$$\tilde{\mathsf{F}}.\mathsf{Ev}(1^\lambda, k, 1^c) = \Pi^{-c}_{tk}(ST_0)$$
$$\tilde{\mathsf{F}}.\mathsf{Con}(1^\lambda, (tk, ST_0), R_n) = (pub, \Pi^{-n}_{tk}(ST_0))$$
$$\tilde{\mathsf{F}}.\mathsf{CEv}(1^\lambda, (pub, ST_n)), R_n, c) = \Pi^{n-c}_{pub}(ST_n).$$

Bost, et al. specifically considered an RSA-based trapdoor permutation and claim, "We can easily reduce the constrained-PRF security of $\tilde{F}$ to the hardness of the RSA assumption, and directly deduce the security of $\Sigma o\phi o\zeta$" [11, p.12, ePrint].

This is incorrect; $\tilde{\mathsf{F}}$ is not secure as a CPRF. Consider the following attack against IND*-NC-CPRF security. The adversary $\mathcal{A}$ first queries $ST_2 \leftarrow \textsc{Ev}(u, 1^2)$. Then it queries $(pub, ST_1) \leftarrow \textsc{Con}(u, R_1)$. It outputs 1 iff $\Pi_{pub}(ST_2) = ST_1$. This adversary always outputs 1 in the real world but will only output 1 in the random world with probably $1/N$, where N is the size of the permutation's range. So its advantage is $1-1/N$. We emphasize that while $\tilde{\mathsf{F}}$ is insecure, it does not mean $\Sigma o\phi o\zeta$ is insecure — just that its security is not captured by the proof flow being proposed by Bost, et al. A separate, direct proof was given by Bost [10] when originally proposing $\Sigma o\phi o\zeta$.

For our purposes, we will show that $\mathsf{Hashed}[\tilde{\mathsf{F}}]$ (which we will call TDCF) is still secure despite $\tilde{\mathsf{F}}$ not being a secure CPRF. The above attack is prevented because the output of Ev is hashed first. All we need of $\tilde{\mathsf{F}}$'s outputs are that they are hard to guess, not that they are indistinguishable from random. This is captured by the following theorem.

Theorem 3. *If the trapdoor permutation is secure with* P *and* P *is consistent and non-programmable, then* TDCF *is SIM*-AC-CPRF secure with* $\mathsf{P} \times \mathsf{P}_{\mathsf{rom}}$.

The proof is given in Appendix D.2 as a modification of the proof of Theorem 2 where we use the security of the trapdoor permutation in the guessing argument to show that the adversary cannot query the random oracle early on values the simulator will later program it at. Because $\mathsf{CF.Inp} = 1^*$ the assumption that adversaries Ev queries are restricted to a set of polynomial size (required for Theorem 2) follows from the adversary being PPT.

5.2 Composed Construction

As discussed above, the searchable encryption scheme Diana [11], essentially, instantiates the CPRF of $\mathsf{Hashed}[\mathsf{CF}]$ using the GGM constrained PRF. The known IND*-NC-CPRF security of GGM plus the analysis of $\mathsf{Hashed}[\mathsf{CF}]$ essentially shows security of Diana. (More formally, using this analysis and our analysis of $\mathsf{FS2}$ gives a security proof for a variant of Diana.)

We want to show that GGM itself is SIM*-AC-CPRF secure, and therefore the extra hash at the end is not needed. We start here by analyzing when composing two secure CPRFs gives a larger secure CPRF. In Section 5.3 we will show that GGM (and related constructions) can be viewed as being obtained from repeated applications of this composition.

In our composed construction, we will use function H to derive a key for the second function F as in $\mathsf{F}(\mathsf{H}(k, x'), x'')$. For our later use, it will be useful to preserve some of the domain of H (called $\mathsf{H.Inp}_1$) for direct use of H without applying F.

Definition 1 (Composed Construction). *Let* F *and* H *be function families with* $\mathsf{H.Inp} = \mathsf{H.Inp}_1 \sqcup \mathsf{H.Inp}_2$ *where* $\mathsf{H.Out}(x) = \mathsf{F.K}$ *for all* $x \in \mathsf{H.Inp}_2$. *Then* $\mathsf{CMP}[\mathsf{F}, \mathsf{H}]$ *has* $\mathsf{CMP}[\mathsf{F}, \mathsf{H}].\mathsf{K} = \mathsf{H.K}$, $\mathsf{CMP}[\mathsf{F}, \mathsf{H}].\mathsf{Inp} = \mathsf{H.Inp}_1 \sqcup (\mathsf{H.Inp}_2 \times \mathsf{F.Inp})$, $\mathsf{CMP}[\mathsf{F}, \mathsf{H}].\mathsf{Out}(x) = \mathsf{H.Out}(x)$ *for* $x \in \mathsf{H.Inp}_1$, $\mathsf{CMP}[\mathsf{F}, \mathsf{H}].\mathsf{Out}((x', x'')) = \mathsf{F.Out}(x'')$ *for* $(x', x'') \in \mathsf{H.Inp}_2 \times \mathsf{F.Inp}$ *and*

$$\mathsf{CMP}[\mathsf{F}, \mathsf{H}].\mathsf{Ev}^{\mathsf{P}}(k, x) = \begin{cases} \mathsf{H.Ev}^{\mathsf{P}}(k, x), & x \in \mathsf{H.Inp}_1 \\ \mathsf{F.Ev}^{\mathsf{P}}(\mathsf{H.Ev}^{\mathsf{P}}(k, x'), x''), & x = (x', x'') \in \mathsf{H.Inp}_2 \times \mathsf{F.Inp}. \end{cases}$$

The use of $\sqcup$ represents a requirement that the union be disjoint.

Suppose that both H and F are constrained function families. Then we can make $\mathsf{CMP}[\mathsf{F},\mathsf{H}]$ into a constrained function family by constraining the key with respect to H or applying H for some input in $\mathsf{H.Inp}_2$ to get a key for F, then constraining that key with F. The first type allows us to constrain $\mathsf{CMP}[\mathsf{F},\mathsf{H}]$ for any sets $R_S = (S \cap \mathsf{H.Inp}_1) \cup ((S \cap \mathsf{H.Inp}_2) \times \mathsf{F.Inp})$ where $S \in \mathsf{H.SS}$. The second type allows us to constrain $\mathsf{CMP}[\mathsf{F},\mathsf{H}]$ for any sets $R_{x,S} = \{x\} \times S$ where $x \in \mathsf{H.Inp}_2$ and $S \in \mathsf{F.SS}$.

More formally, the relevant algorithms of $\mathsf{CMP}[\mathsf{F},\mathsf{H}]$ are as follows.

$\underline{\mathsf{CMP}[\mathsf{F},\mathsf{H}].\mathsf{Con}^{\mathsf{P}}(1^\lambda, k, R)}$
If $R = R_S$:
  $K_R \leftarrow \mathsf{H.Con}^{\mathsf{P}}(1^\lambda, k, S)$
Else if $R = R_{x,S}$:
  $K_x \leftarrow \mathsf{H.Ev}^{\mathsf{P}}(1^\lambda, k, x)$
  $K_R \leftarrow \mathsf{F.Con}^{\mathsf{P}}(1^\lambda, K_x, S)$
Return K_R

$\underline{\mathsf{CMP}[\mathsf{F},\mathsf{H}].\mathsf{CEv}^{\mathsf{P}}(1^\lambda, K_R, R, \boldsymbol{x})}$
If $R = R_S$ and $\boldsymbol{x} \in \mathsf{H.Inp}_1$:
  $y \leftarrow \mathsf{H.CEv}^{\mathsf{P}}(1^\lambda, K_R, S, \boldsymbol{x})$
Else if $R = R_S$ and $\boldsymbol{x} = (x, x')$:
  $K_x \leftarrow \mathsf{H.CEv}^{\mathsf{P}}(1^\lambda, K_R, S, x)$
  $y \leftarrow \mathsf{F.Ev}^{\mathsf{P}}(1^\lambda, K_x, x')$
Else if $R = R_{x,S}$ and $\boldsymbol{x} = (x, x')$:
  $y \leftarrow \mathsf{F.CEv}^{\mathsf{P}}(1^\lambda, K_R, S, x')$
Return y

Security Claims. Before giving our proof we need a technical consistency property that we will assume of F and H.

Definition 2. *Constrained function family* F *is* strongly consistent *with* P *if* $K_R \leftarrow \mathsf{F.Con}^{\mathsf{P}}(k, R)$ *was computed, then all future computations* $y \leftarrow \mathsf{F.Ev}^{\mathsf{P}}(k, x)$ *or* $y \leftarrow \mathsf{F.CEv}^{\mathsf{P}}(K_R, R, x)$ *for* $x \in R$ *give the same* y*, no matter what queries to* PPrim *are made in between, and the* P *queries made by* $\mathsf{F.Con}$ *and* $\mathsf{F.CEv}$ *match those made by* $\mathsf{F.Ev}$*.*

The constrained function families considered in our work naturally satisfy this if P is consistent. These conditions allow evaluations with k to be undetectably replaced with constrained evaluations by an already computed K_R. The following theorem captures the security we can prove of the composed construction.

Theorem 4. *For* $X \in \{$*IND*-NC, SIM*-AC*$\}$*, if* F *and* H *are X-CPRF secure and strongly consistent with* P*, then so is* $\mathsf{CMP}[\mathsf{F},\mathsf{H}]$*.*

Details of the proof are in Appendix E.1 (IND*-NC) and Appendix E.2 (SIM*-AC). The "natural" proof which applies security of H then security of F works for IND*-NC security (and does not actually require strong consistency). We replace all keys H outputs during Ev queries with random. For Con queries $R_{x,S}$ where x was used in previous Ev queries we leave the output of H random. For other Con queries we used H honestly. Then we apply security of F to replace its output with random for Ev queries, using that the underlying keys were switched to random in the previous step.

This proof idea breaks down if we try to replace IND*-NC with IND*-AC. The issue arises when the attacker makes Con queries of the form $R_{x,S}$. For this query, we want to switch $K_x = \mathsf{H.Ev}^{\mathsf{P}}(K, x)$ to a random value if the attacker later makes an Ev query of the form (x, x''). We want to leave K_x nonrandom if the attacker is going to later make a Con query for R_S where $x \in S$. With IND*-NC security, all Ev queries had to be made first, so we knew whether the first case would occur by the time the Con query was made. With IND*-AC security, queries can occur in arbitrary order, so we do not know which case will occur.

With SIM*-AC security, we can again resolve the issue. We can switch K_x to random, then later if a Con query is made which should cause K_x to look nonrandom we use the simulator for H to program the ideal primitive appropriately. However, SIM*-AC security also creates a "race condition" between the simulators of F and H. Intuitively, when a constrain query has been made on set $S \subseteq \mathsf{H.Inp}_1$, the simulator for H will pick the constrained key (and thus "owns" all inputs in S). Later, when another constrain query is made on $x \times \mathsf{F.Inp}$ with $x \in S \cap \mathsf{H.Inp}_2$, the game has to execute $\mathsf{H.CEv}$ and $\mathsf{F.Con}$ to derive the actual output, as the adversary has seen the constrain key. Conversely, when the $x \times \mathsf{F.Inp}$ is made first, the simulator for F will pick the key (and thus "owns" x). If a query is later made on $S \subseteq \mathsf{H.Inp}_1$ with $x \in S$, then the H simulator needs to be informed of the key picked by the F simulator so that it can program the ideal primitive appropriately. The H simulator will then take over "ownership" of x. In the proof we need to carefully track the ownership of inputs to simulate correctly. We give the proof in Appendix E.2.

$\underline{\mathsf{S.Init}(1^\lambda)}$
$\sigma_\mathsf{H} \leftarrow\!\!\$\ \mathsf{S_H.Init}(1^\lambda)$
$\sigma_\mathsf{F} \leftarrow\!\!\$\ \mathsf{S_F.Init}(1^\lambda)$
$\mathtt{CKey} \leftarrow [\cdot]$
$\mathtt{Own} \leftarrow [\cdot]$
Return $(\sigma_\mathsf{H}, \sigma_\mathsf{F}, \mathtt{CKey}, \mathtt{Own})$

$\underline{\mathsf{S.Ev}^{\mathrm{PPRIM}}(1^\lambda, u, \boldsymbol{x} : \sigma)}$
$(K_R, R) \leftarrow \mathtt{CKey}_u[\boldsymbol{x}]$
$y \leftarrow \mathsf{CMP.CEv}^\mathsf{P}(K_R, R, \boldsymbol{x})$
Return y

$\underline{\mathsf{S.Con}^{\mathrm{PPRIM}}(1^\lambda, u, R, f' : \sigma)}$
$f^\mathsf{H}, f^\mathsf{F} \leftarrow [\cdot]$
For $\boldsymbol{x} \in f'.\mathsf{D} \cap \mathsf{H.Inp}_1$:
  $f^\mathsf{H}_u(\boldsymbol{x}) \leftarrow f'(\boldsymbol{x})$
For $(x, x') \in f'.\mathsf{D} \cap (\mathsf{H.Inp}_2 \times \mathsf{F.Inp})$:
  $f^\mathsf{F}_{u,x}(x') \leftarrow f'(x, x')$
If $R = R_S$:
  For $x \in S$ s.t. $\mathtt{Own}_u[x] = \mathsf{F}$ or $f^\mathsf{F}_{u,x}.\mathsf{D} \neq \emptyset$:
    $f^\mathsf{H}_u(x) \leftarrow \mathsf{S_F.Con}^{\mathrm{PPRIM}}((u, x), \mathsf{F.Inp}, f^\mathsf{F}_{u,x} : \sigma_\mathsf{F})$
  $K_R \leftarrow \mathsf{S_H.Con}^{\mathrm{PPRIM}}(1^\lambda, u, S, f^\mathsf{H}_u : \sigma_\mathsf{H})$
  $\mathtt{CKey}_u.\mathsf{store}(S \cap \mathsf{H.Inp}_2 \mapsto (K_R, S))$
  $\mathtt{Own}_u.\mathsf{store}(S \cap \mathsf{H.Inp}_2 \mapsto \mathsf{H})$
Else If $R = R_{x,S}$:
  $\mathsf{X} \leftarrow \mathtt{Own}_u[x]$
  If $\mathsf{X} \in \{\bot, \mathsf{F}\}$:
    $K_R \leftarrow \mathsf{S_F.Con}^{\mathrm{PPRIM}}(1^\lambda, (u, x), S, f^\mathsf{F}_{u,x} : \sigma_\mathsf{F})$
    $\mathtt{Own}_u.\mathsf{store}(\{x\} \mapsto \mathsf{F})$
  Else:
    $(K_{S'}, S') \leftarrow \mathtt{CKey}_u[x]$
    $K_x \leftarrow \mathsf{H.CEv}^\mathsf{P}(1^\lambda, K_{S'}, S', x)$
    $K_R \leftarrow \mathsf{F.Con}^\mathsf{P}(1^\lambda, K_x, S)$
$\mathtt{CKey}_u.\mathsf{store}(R \mapsto (K_R, R))$
Return K_R

Fig. 5. Simulator S for the SIM*-AC-CPRF security proof of $\mathsf{CMP}[\mathsf{F}, \mathsf{H}]$ (Theorem 4). A $T.\mathsf{store}(S \mapsto \mathsf{X})$ operation *lazily* assigns $T[x] = \mathsf{X}$ for all x in S. It uses table $\mathtt{Own}$ to track which simulator "owns" different the F keys. Ownership transfers from F to H when keys output by $\mathsf{S_F}$ are given to $\mathsf{S_H}$. It uses table $\mathtt{CKey}$ to track previously returned constrained keys for use in emulating future evaluations of CMP or H using constrained evaluations. Correctness of this uses the strong consistency of H and F.

The simulator we use in the proof is shown in Fig. 5. Tables $\mathtt{Own}$ tracker the owner of x. Strong consistency is needed so that S can use previously returned constrained keys to simulate evaluations by performing constrained evaluations instead (using the table $\mathtt{CKey}$ to track constrained keys). Inside $\mathsf{S.Con}$ the partial function f' is used to reproduce any partial functions required by $\mathsf{S_F.Con}$. Both f' and $\mathsf{S_F.Con}$ are used to reproduce partial functions required by $\mathsf{S_H.Con}$. The low-level details of S are best understood by reading the proof which uses eight hybrid games (mixing transitions based on the the security of H and F with transitions that are syntactic rewritings) to slowly transform from the real SIM*-AC-CPRF world to a game matching the simulated world when S is used.

5.3 Cascade-based Constructions

Cascade Construction. Bellare, Canetti, and Krawczyk [6] studied the cascade construction of a PRF which iterates a function family for small inputs to construct one for that accepts large inputs. We formalize it as following:

Definition 3 (Cascade Construction [6]). *Let* F *be a function family with* $\mathsf{F.Out} = \mathsf{F.K}$. *Let* D *be a subset of* $\mathsf{F.Inp}^*$. *Then* $\mathsf{CSC}[\mathsf{F}, \mathsf{D}]$ *has* $\mathsf{CSC}[\mathsf{F}, \mathsf{D}].\mathsf{K} = \mathsf{F.K}$, $\mathsf{CSC}[\mathsf{F}, \mathsf{D}].\mathsf{Inp} = \mathsf{D}$, $\mathsf{CSC}[\mathsf{F}, \mathsf{D}].\mathsf{Out} = \mathsf{F.K}$, *and evaluation function* $\mathsf{CSC}[\mathsf{F}, \mathsf{D}].\mathsf{Ev}^\mathsf{P}(1^\lambda, k, (x_0, \ldots, x_{n-1}))$: *"For* $i = 0, \ldots, n-1$ *do* $k \leftarrow \mathsf{F.Ev}^\mathsf{P}(1^\lambda, k, x_i)$*; Return* k*".*

We sometimes want to run this evaluation function without regard to any particular choice of D, in which case we will simply write $\mathsf{CSC}[\mathsf{F}].\mathsf{Ev}$.

Note that $\mathsf{CSC}[\mathsf{F}, \mathsf{D}]$ is trivially insecure if any element of D is a prefix of another element due to length-extension attacks. If $\boldsymbol{x} \sqsubseteq \boldsymbol{y}$, then the output of CSC on $\boldsymbol{y}$ can easily be computed from the output of CSC on

$\boldsymbol{x}$. A PRF adversary can easily achieve advantage $1-1/|\mathsf{F.Out}|$ by making queries for $\boldsymbol{x}$ and $\boldsymbol{y}$ then checking if the output of the latter query matches the output it can compute itself from the former query. Because of this we will restrict attention to "prefix-free" sets D for which no element is a prefix of another. (We additionally require the technical assumption that for a given $\boldsymbol{x}$ we can efficiently check whether $\exists \boldsymbol{y} \in \mathsf{D}, \boldsymbol{x} \sqsubseteq \boldsymbol{y}$ holds.) The most common choice is $\mathsf{D} = \mathsf{F.Inp}^n$.

The same property allows CSC to be used as a prefix-constrained function family. We define

- $\mathsf{CSC}[\mathsf{F},\mathsf{D}].\mathsf{SS} = \{R_{x\sqsubseteq} \mid x \in \mathsf{F.Inp}^*\}$,
- $\mathsf{CSC}[\mathsf{F},\mathsf{D}].\mathsf{Con}^{\mathsf{P}}(1^\lambda, k, R_{x\sqsubseteq}) = \mathsf{CSC}[\mathsf{F}].\mathsf{Ev}^{\mathsf{P}}(1^\lambda, k, x)$, and
- $\mathsf{CSC}[\mathsf{F},\mathsf{D}].\mathsf{CEv}^{\mathsf{P}}(1^\lambda, K, R_{x\sqsubseteq}, y) = \mathsf{CSC}[\mathsf{F}].\mathsf{Ev}^{\mathsf{P}}(1^\lambda, K, y \setminus x)$.

Here $y \setminus x$ denotes the suffix of y starting after the end of x.

For a given F with $\mathsf{F.Inp} = \{0,1\}$, define G by $G(k) = \mathsf{F}(k,0) \circ \mathsf{F}(k,1)$. Note that G is a length-doubling function. By a straightforward analysis, F is a secure PRF if and only if G is a secure PRG. As noted by Bellare, Canetti, and Krawczyk, the function family obtained by applying the cascade construction to F is identical to the GGM construction of a PRF [19] applied to G. Thus CSC can be viewed as a strict generalization of GGM.

Augmented Cascade Construction. Bellare, Bernstein, and Tessaro [5] considered a modification to the cascade to allow it to be used securely for domain $\mathsf{F.Inp}^*$, rather than having to restrict to a prefix-free domain. All output values are first shrunk by applying a keyless function Chop. Intuitively, shrinking the output sufficiently should prevent the length-extension attacks described above because the internal key value cannot be recovered from this shrunken output.

Definition 4 (Augmented Cascade [5]). *Let* F *be a function family with* $\mathsf{F.Out} = \mathsf{F.K}$. *Let* Chop *be a keyless function which outputs values in* $\mathsf{Chop.Out}$ *given inputs in* $\mathsf{F.K}$. *then* $\mathsf{ACSC}[\mathsf{Chop},\mathsf{F}]$ *has keyspace* $\mathsf{ACSC}[\mathsf{Chop},\mathsf{F}].\mathsf{K} = \mathsf{F.K}$, *domain* $\mathsf{ACSC}[\mathsf{Chop},\mathsf{F}].\mathsf{Inp} = \mathsf{F.Inp}^*$, *range* $\mathsf{ACSC}[\mathsf{Chop},\mathsf{F}].\mathsf{Out} = \mathsf{Chop.Out}$, *and evaluation function defined by* $\mathsf{ACSC}[\mathsf{Chop},\mathsf{F}].\mathsf{Ev}^{\mathsf{P}}(1^\lambda, k, \boldsymbol{x})$: *"*$k \leftarrow \mathsf{CSC}[\mathsf{F}].\mathsf{Ev}^{\mathsf{P}}(1^\lambda, k, \boldsymbol{x})$; *Return* $\mathsf{Chop}(1^\lambda, k)$*".*

$\mathsf{ACSC}[\mathsf{Chop},\mathsf{F}]$ can be viewed as a prefix-constrained PRF analogously to CSC. We define

- $\mathsf{ACSC}[\mathsf{Chop},\mathsf{F}].\mathsf{SS} = \{R_{x\sqsubseteq} \mid x \in \mathsf{F.Inp}^*\}$,
- $\mathsf{ACSC}[\mathsf{Chop},\mathsf{F}].\mathsf{Con}^{\mathsf{P}}(1^\lambda, k, R_{x\sqsubseteq}) = \mathsf{CSC}[\mathsf{F}].\mathsf{Ev}^{\mathsf{P}}(1^\lambda, k, x)$, and
- $\mathsf{ACSC}[\mathsf{Chop},\mathsf{F}].\mathsf{CEv}^{\mathsf{P}}(1^\lambda, K, R_{x\sqsubseteq}, y) = \mathsf{ACSC}[\mathsf{Chop},\mathsf{F}].\mathsf{Ev}^{\mathsf{P}}(1^\lambda, K, y \setminus x)$.

Results of Bellare, Bernstein, and Tessaro [5] imply that this is a secure PRF assuming that F is a secure PRF even against attackers given leakage of Chop applied to the key. (Recall this is equivalent to assuming that $\mathsf{F}_{\mathsf{Chop}}$ is a secure PRF where $\mathsf{F}_{\mathsf{Chop}}$ is defined to be identical to F except that $\mathsf{F}_{\mathsf{Chop}}.\mathsf{Ev}^{\mathsf{P}}(1^\lambda, K, \varepsilon) = \mathsf{Chop}(1^\lambda, K)$ where $\varepsilon \notin \mathsf{F.Inp}$.)

Security Claim. The following captures our security results for CSC and ACSC.

Theorem 5. *If* F *is SIM*-AC-PRF secure with* P *and* D *is prefix-free, then* $\mathsf{CSC}[\mathsf{F},\mathsf{D}]$ *is SIM*-AC-CPRF secure with* P. *If* $\mathsf{F}_{\mathsf{Chop}}$ *is SIM*-AC-PRF secure with* P, *then* $\mathsf{ACSC}[\mathsf{Chop},\mathsf{F}]$ *is SIM*-AC-CPRF secure with* P. *The same holds with IND*-PRF security of* F *(resp.* $\mathsf{F}_{\mathsf{Chop}}$*) and IND*-NC-CPRF security of* $\mathsf{CSC}[\mathsf{F},\mathsf{D}]$ *(resp.* $\mathsf{ACSC}[\mathsf{Chop},\mathsf{F}]$*).*

Proof (Sketch). Consider $\mathsf{D} = \mathsf{F.Inp}^n$. Recursively define $\mathsf{CSC}_1 = \mathsf{F}$ (viewing F as a constrained function family) and $\mathsf{CSC}_{i+1} = \mathsf{CMP}[\mathsf{F}, \mathsf{CSC}_i]$. Setting $\mathsf{CSC_i.Inp}_1 = \emptyset$ and $\mathsf{CSC_i.Inp}_2 = \mathsf{F.Inp}^i$ gives $\mathsf{CSC}_{i+1}.\mathsf{Inp} = \mathsf{F.Inp}^i$. We claim that $\mathsf{CSC}_n = \mathsf{CSC}[\mathsf{F},\mathsf{D}]$ and thus the claim follows by applying Theorem 4 $n-1$ times (using Proposition 1 for the SIM*-AC-CPRF security of F).

Augmented cascade works similarly by setting $\mathsf{ACSC}_0 = \mathsf{F}_{\mathsf{Chop}}$ and $\mathsf{ACSC}_{i+1} = \mathsf{CMP}[\mathsf{F}_{\mathsf{Chop}}, \mathsf{ACSC}_i]$. Setting $\mathsf{ACSC_i.Inp}_1 = \mathsf{F.Inp}^{\le i}$ and $\mathsf{ACSC_i.Inp}_2 = \mathsf{F.Inp}^{i+1}$ gives

$$\mathsf{ACSC_{i+1}.Inp} = \mathsf{F.Inp}^{\le i} \sqcup \mathsf{F.Inp}^{i+1} \times (\{\varepsilon\} \sqcup \mathsf{F.Inp}) = \mathsf{F.Inp}^{\le i+2}.$$

Here we mildly abuse notation to identify the set $\mathsf{F.Inp}^{i+1} \times \{\varepsilon\}$ with the set $\mathsf{F.Inp}^{i+1}$ in the natural way.

If the adversary makes queries of length at most n (meaning queries always in $\mathsf{F.Inp}^{\leq n}$), then ACSC_n will behave equivalently to $\mathsf{ACSC}[\mathsf{Chop}, \mathsf{F}]$ and thus the claim follows by applying Theorem 4 $n-1$ times (using Proposition 1 for the SIM*-AC-CPRF security of $\mathsf{F}_{\mathsf{Chop}}$). □

The full proof is in Appendix F. It proves security for any prefix-free D. Moreover, a literal use of the above would only work for a constant value of n. To handle arbitrary polynomial n we use details of the actual proof of Theorem 4, in particular that its simulator and reductions adversaries are sufficiently blackbox and uniform. We use this to build a recursive simulator and recursive reduction adversaries that work without having to a priori fix n. The ideas sketched above can then be applied using these adversaries.

5.4 Augmented MAC

In general, the augmented MAC hash function AMAC is defined by $\mathsf{Chop}(H(K \circ M))$ for a hash function H. An important special case, analyzed by Bellare, Bernstein, and Tessaro [5], is when H is a MD-style hash function like SHA512 and $|K|$ is the blocklength of the underlying compression function. Then the evaluation of AMAC on k and input $(x_0, \ldots, x_{n-1})$ computes $k \leftarrow \mathsf{F}(s, k)$ and then $k \leftarrow \mathsf{F}(k, x_i)$ for $i = 0, \ldots, n-1$ before returning $\mathsf{Chop}(k)$. Here s is the seed value of the hash function.

This is basically an instance of ACSC, except the inputs to the first call of F were flipped. If F is a function family, we will let F' denote its dual defined by $\mathsf{F}'.\mathsf{Inp} = \mathsf{F.K}$, $\mathsf{F}'.\mathsf{Out} = \mathsf{F.Out}$, $\mathsf{F}'.\mathsf{K} = \mathsf{F.Inp}$, and $\mathsf{F}'.\mathsf{Ev}(k, x) = \mathsf{F.Ev}(x, k)$. Then we can capture this special case of ACSC as follows (where we consider the seed as part of the input, rather than modeling it as a fixed value).

Definition 5 (Augmented MAC [5]). *We define* $\mathsf{AMAC}[\mathsf{Chop}, \mathsf{F}] = \mathsf{CMP}[\mathsf{ACSC}[\mathsf{Chop}, \mathsf{F}], \mathsf{F}']$ *with* $\mathsf{F}'.\mathsf{Inp}_1 = \emptyset$ *and* $\mathsf{F}'.\mathsf{Inp}_2 = \mathsf{F}'.\mathsf{Inp}$*, for appropriate* F *and* Chop.

The domain of this function is $\mathsf{F.K} \times \mathsf{F.Inp}^*$. It is a prefixed-constrained function family. The following captures its security as a corollary of our earlier results.

Corollary 1. *If* $\mathsf{F}_{\mathsf{Chop}}$ *and* F' *are both SIM*-AC-PRF secure with primitive* P*, then* $\mathsf{AMAC}[\mathsf{Chop}, \mathsf{F}]$ *is SIM*-AC-CPRF secure with* P*. The same holds with IND*-NC security.*

Proof. From Theorem 5, we have that $\mathsf{ACSC}[\mathsf{Chop}, \mathsf{F}]$ is secure with P. By Proposition 1, security of F' as a function family implies security of it as a constrained function family. Hence Theorem 4 implies the security of $\mathsf{CMP}[\mathsf{ACSC}[\mathsf{Chop}, \mathsf{F}], \mathsf{F}']$. □

Acknowledgements We thank Nirvan Tyagi, whose recommendation we examine how SIM-AC relates to the settings in [24] inspired this work.

References

1. Aviram, N., Gellert, K., Jager, T.: Session resumption protocols and efficient forward security for TLS 1.3 0-RTT. Journal of Cryptology **34**(3), 20 (Jul 2021). https://doi.org/10.1007/s00145-021-09385-0 2, 12
2. Backendal, M., Günther, F., Paterson, K.G.: Puncturable key wrapping and its applications. In: Agrawal, S., Lin, D. (eds.) ASIACRYPT 2022, Part II. LNCS, vol. 13792, pp. 651–681. Springer, Cham, Switzerland, Taipei, Taiwan (Dec 5–9, 2022). https://doi.org/10.1007/978-3-031-22966-4_22 2, 3, 11, 12, 28, 29, 30
3. Baum, C., Beullens, W., Mukherjee, S., Orsini, E., Ramacher, S., Rechberger, C., Roy, L., Scholl, P.: One tree to rule them all: Optimizing GGM trees and OWFs for post-quantum signatures. In: Chung, K.M., Sasaki, Y. (eds.) ASIACRYPT 2024, Part I. LNCS, vol. 15484, pp. 463–493. Springer, Singapore, Singapore, Kolkata, India (Dec 9–13, 2024). https://doi.org/10.1007/978-981-96-0875-1_15 2
4. Baum, C., Braun, L., Delpech de Saint Guilhem, C., Klooß, M., Orsini, E., Roy, L., Scholl, P.: Publicly verifiable zero-knowledge and post-quantum signatures from VOLE-in-the-head. In: Handschuh, H., Lysyanskaya, A. (eds.) CRYPTO 2023, Part V. LNCS, vol. 14085, pp. 581–615. Springer, Cham, Switzerland, Santa Barbara, CA, USA (Aug 20–24, 2023). https://doi.org/10.1007/978-3-031-38554-4_19 2

5. Bellare, M., Bernstein, D.J., Tessaro, S.: Hash-function based PRFs: AMAC and its multi-user security. In: Fischlin, M., Coron, J.S. (eds.) EUROCRYPT 2016, Part I. LNCS, vol. 9665, pp. 566–595. Springer Berlin Heidelberg, Germany, Vienna, Austria (May 8–12, 2016). https://doi.org/10.1007/978-3-662-49890-3_22 2, 4, 6, 7, 17, 18, 49, 50, 52, 55
6. Bellare, M., Canetti, R., Krawczyk, H.: Pseudorandom functions revisited: The cascade construction and its concrete security. In: 37th FOCS. pp. 514–523. IEEE Computer Society Press, Burlington, Vermont (Oct 14–16, 1996). https://doi.org/10.1109/SFCS.1996.548510 2, 3, 16
7. Bellare, M., Davis, H., Di, Z.: Hardening signature schemes via derive-then-derandomize: Stronger security proofs for EdDSA. In: Boldyreva, A., Kolesnikov, V. (eds.) PKC 2023, Part I. LNCS, vol. 13940, pp. 223–250. Springer, Cham, Switzerland, Atlanta, GA, USA (May 7–10, 2023). https://doi.org/10.1007/978-3-031-31368-4_9 49
8. Bellare, M., Rogaway, P.: The security of triple encryption and a framework for code-based game-playing proofs. In: Vaudenay, S. (ed.) EUROCRYPT 2006. LNCS, vol. 4004, pp. 409–426. Springer Berlin Heidelberg, Germany, St. Petersburg, Russia (May 28 – Jun 1, 2006). https://doi.org/10.1007/11761679_25 4, 35
9. Boneh, D., Waters, B.: Constrained pseudorandom functions and their applications. In: Sako, K., Sarkar, P. (eds.) ASIACRYPT 2013, Part II. LNCS, vol. 8270, pp. 280–300. Springer Berlin Heidelberg, Germany, Bengalore, India (Dec 1–5, 2013). https://doi.org/10.1007/978-3-642-42045-0_15 2, 3, 7, 9, 11, 13
10. Bost, R.: $\Sigma o \phi o \varsigma$: Forward secure searchable encryption. In: Weippl, E.R., Katzenbeisser, S., Kruegel, C., Myers, A.C., Halevi, S. (eds.) ACM CCS 2016. pp. 1143–1154. ACM Press, Vienna, Austria (Oct 24–28, 2016). https://doi.org/10.1145/2976749.2978303 2, 10, 13, 14
11. Bost, R., Minaud, B., Ohrimenko, O.: Forward and backward private searchable encryption from constrained cryptographic primitives. In: Thuraisingham, B.M., Evans, D., Malkin, T., Xu, D. (eds.) ACM CCS 2017. pp. 1465–1482. ACM Press, Dallas, TX, USA (Oct 31 – Nov 2, 2017). https://doi.org/10.1145/3133956.3133980 2, 3, 10, 11, 12, 13, 14, 21, 22
12. Boyle, E., Goldwasser, S., Ivan, I.: Functional signatures and pseudorandom functions. In: Krawczyk, H. (ed.) PKC 2014. LNCS, vol. 8383, pp. 501–519. Springer Berlin Heidelberg, Germany, Buenos Aires, Argentina (Mar 26–28, 2014). https://doi.org/10.1007/978-3-642-54631-0_29 2, 3, 7, 9, 11, 13
13. Camenisch, J., Drijvers, M., Gagliardoni, T., Lehmann, A., Neven, G.: The wonderful world of global random oracles. In: Nielsen, J.B., Rijmen, V. (eds.) EUROCRYPT 2018, Part I. LNCS, vol. 10820, pp. 280–312. Springer, Cham, Switzerland, Tel Aviv, Israel (Apr 29 – May 3, 2018). https://doi.org/10.1007/978-3-319-78381-9_11 2, 5
14. Cash, D., Jaeger, J., Jarecki, S., Jutla, C.S., Krawczyk, H., Rosu, M.C., Steiner, M.: Dynamic searchable encryption in very-large databases: Data structures and implementation. In: NDSS 2014. The Internet Society, San Diego, CA, USA (Feb 23–26, 2014). https://doi.org/10.14722/ndss.2014.23264 10
15. Cheng, K., Jaeger, J.: Adaptive security for constrained PRFs. In: Kalai, Y.T., Kamara, S.F. (eds.) CRYPTO 2025, Part V. LNCS, vol. 16004, pp. 598–627. Springer, Cham, Switzerland, Santa Barbara, CA, USA (Aug 17–21, 2025). https://doi.org/10.1007/978-3-032-01901-1_19 1
16. Curtmola, R., Garay, J.A., Kamara, S., Ostrovsky, R.: Searchable symmetric encryption: improved definitions and efficient constructions. In: Juels, A., Wright, R.N., De Capitani di Vimercati, S. (eds.) ACM CCS 2006. pp. 79–88. ACM Press, Alexandria, Virginia, USA (Oct 30 – Nov 3, 2006). https://doi.org/10.1145/1180405.1180417 10
17. Du, M., Wu, S., Wang, Q., Chen, D., Jiang, P., Mohaisen, A.: Graphshield: Dynamic large graphs for secure queries with forward privacy. IEEE Transactions on Knowledge and Data Engineering **34**(7), 3295–3308 (2020) 2, 10, 11
18. Fuchsbauer, G., Konstantinov, M., Pietrzak, K., Rao, V.: Adaptive security of constrained PRFs. In: Sarkar, P., Iwata, T. (eds.) ASIACRYPT 2014, Part II. LNCS, vol. 8874, pp. 82–101. Springer Berlin Heidelberg, Germany, Kaoshiung, Taiwan, R.O.C. (Dec 7–11, 2014). https://doi.org/10.1007/978-3-662-45608-8_5 3, 7, 13
19. Goldreich, O., Goldwasser, S., Micali, S.: How to construct random functions (extended abstract). In: 25th FOCS. pp. 464–479. IEEE Computer Society Press, Singer Island, Florida (Oct 24–26, 1984). https://doi.org/10.1109/SFCS.1984.715949 2, 3, 7, 11, 17
20. Green, M.D., Miers, I.: Forward secure asynchronous messaging from puncturable encryption. In: 2015 IEEE Symposium on Security and Privacy. pp. 305–320. IEEE Computer Society Press, San Jose, CA, USA (May 17–21, 2015). https://doi.org/10.1109/SP.2015.26 12
21. Hofheinz, D., Kastner, J., Klein, K.: The power of undirected rewindings for adaptive security. In: Handschuh, H., Lysyanskaya, A. (eds.) CRYPTO 2023, Part II. LNCS, vol. 14082, pp. 725–758. Springer, Cham, Switzerland, Santa Barbara, CA, USA (Aug 20–24, 2023). https://doi.org/10.1007/978-3-031-38545-2_24 3, 7, 13
22. Jaeger, J.: Let attackers program ideal models: Modularity and composability for adaptive compromise. In: Hazay, C., Stam, M. (eds.) EUROCRYPT 2023, Part III. LNCS, vol. 14006, pp. 101–131. Springer, Cham, Switzerland, Lyon, France (Apr 23–27, 2023). https://doi.org/10.1007/978-3-031-30620-4_4 2, 5, 6, 21, 22, 50

23. Jaeger, J., Tyagi, N.: Handling adaptive compromise for practical encryption schemes. In: Micciancio, D., Ristenpart, T. (eds.) CRYPTO 2020, Part I. LNCS, vol. 12170, pp. 3–32. Springer, Cham, Switzerland, Santa Barbara, CA, USA (Aug 17–21, 2020). https://doi.org/10.1007/978-3-030-56784-2_1 2, 5, 6, 9, 10, 12, 21, 22, 24
24. Kamath, C., Klein, K., Pietrzak, K., Walter, M.: The cost of adaptivity in security games on graphs. In: Nissim, K., Waters, B. (eds.) TCC 2021, Part II. LNCS, vol. 13043, pp. 550–581. Springer, Cham, Switzerland, Raleigh, NC, USA (Nov 8–11, 2021). https://doi.org/10.1007/978-3-030-90453-1_19 3, 18
25. Kiayias, A., Papadopoulos, S., Triandopoulos, N., Zacharias, T.: Delegatable pseudorandom functions and applications. In: Sadeghi, A.R., Gligor, V.D., Yung, M. (eds.) ACM CCS 2013. pp. 669–684. ACM Press, Berlin, Germany (Nov 4–8, 2013). https://doi.org/10.1145/2508859.2516668 2, 3, 7, 9, 10, 11, 13
26. Nielsen, J.B.: Separating random oracle proofs from complexity theoretic proofs: The non-committing encryption case. In: Yung, M. (ed.) CRYPTO 2002. LNCS, vol. 2442, pp. 111–126. Springer Berlin Heidelberg, Germany, Santa Barbara, CA, USA (Aug 18–22, 2002). https://doi.org/10.1007/3-540-45708-9_8 2, 9
27. Rogaway, P.: Authenticated-encryption with associated-data. In: Atluri, V. (ed.) ACM CCS 2002. pp. 98–107. ACM Press, Washington, DC, USA (Nov 18–22, 2002). https://doi.org/10.1145/586110.586125 12
28. Rogaway, P., Shrimpton, T.: A provable-security treatment of the key-wrap problem. In: Vaudenay, S. (ed.) EUROCRYPT 2006. LNCS, vol. 4004, pp. 373–390. Springer Berlin Heidelberg, Germany, St. Petersburg, Russia (May 28 – Jun 1, 2006). https://doi.org/10.1007/11761679_23 12, 28
29. Sahai, A., Waters, B.: How to use indistinguishability obfuscation: deniable encryption, and more. In: Shmoys, D.B. (ed.) 46th ACM STOC. pp. 475–484. ACM Press, New York, NY, USA (May 31 – Jun 3, 2014). https://doi.org/10.1145/2591796.2591825 2
30. Song, X., Zheng, Y., Bai, J., Dong, C., Liu, Z., Chang, E.C.: DISCO: Dynamic searchable encryption with constant state. In: Zhou, J., Quek, T.Q.S., Gao, D., Cárdenas, A.A. (eds.) ASIACCS 24. ACM Press, Singapore (Jul 1–5, 2024). https://doi.org/10.1145/3634737.3637674 2, 10, 11, 12
31. Sun, S., Yuan, X., Liu, J.K., Steinfeld, R., Sakzad, A., Vo, V., Nepal, S.: Practical backward-secure searchable encryption from symmetric puncturable encryption. In: Lie, D., Mannan, M., Backes, M., Wang, X. (eds.) ACM CCS 2018. pp. 763–780. ACM Press, Toronto, ON, Canada (Oct 15–19, 2018). https://doi.org/10.1145/3243734.3243782 2, 10
32. Tyagi, N., Mughees, M.H., Ristenpart, T., Miers, I.: BurnBox: Self-revocable encryption in a world of compelled access. In: Enck, W., Felt, A.P. (eds.) USENIX Security 2018. pp. 445–461. USENIX Association, Baltimore, MD, USA (Aug 15–17, 2018), https://www.usenix.org/conference/usenixsecurity18/presentation/tyagi 5
33. Wang, J., Sun, S.F., Li, T., Qi, S., Chen, X.: Practical volume-hiding encrypted multi-maps with optimal overhead and beyond. In: Yin, H., Stavrou, A., Cremers, C., Shi, E. (eds.) ACM CCS 2022. pp. 2825–2839. ACM Press, Los Angeles, CA, USA (Nov 7–11, 2022). https://doi.org/10.1145/3548606.3559345 2, 10, 11
34. Yoneyama, K., Kimura, S.: Verifiable and forward secure dynamic searchable symmetric encryption with storage efficiency. In: Qing, S., Mitchell, C., Chen, L., Liu, D. (eds.) ICICS 17. LNCS, vol. 10631, pp. 489–501. Springer, Cham, Switzerland, Beijing, China (Dec 6–8, 2017). https://doi.org/10.1007/978-3-319-89500-0_42 2, 10, 11

A Proof of Theorem 1 (SIM*-AC-CPRF implies IND*-AC-CPRF)

Proof. To show SIM*-AC-CPRF security implies IND*-AC-CPRF security, we build $\mathcal{A}_{\mathsf{cprf}^*}$ using $\mathcal{A}_{\mathsf{cprf}}$ below. We refer to the game $\mathsf{G}^{\mathsf{ind}^*\text{-ac-cprf}}$ that $\mathcal{A}_{\mathsf{cprf}^*}$ is emulating for $\mathcal{A}_{\mathsf{cprf}}$ as the "emulated game", and the game $\mathsf{G}^{\mathsf{sim}^*\text{-ac-cprf}}$ (defined in bottom of Fig. 2) that $\mathcal{A}_{\mathsf{cprf}^*}$ itself is playing as the "hidden game".

$$\frac{\text{Adversary } \mathcal{A}_{\mathsf{cprf}^*}^{\text{PPRIM,CON,EV}}(1^\lambda)}{f_{(\cdot)} \leftarrow\!\!\$\ \mathsf{CF.Fcs}(\lambda)\\ b \leftarrow\!\!\$\ \{0,1\}\\ b' \leftarrow \mathcal{A}_{\mathsf{cprf}}^{\text{PPRIM,CONSIM,EVSIM}}(1^\lambda)\\ \text{Return } (b = b')}$$

$$\frac{\textsc{ConSim}(u, R)}{\text{Require } R \cap E_u = \emptyset\\ K_R \leftarrow \textsc{Con}(u, R)\\ C_u \leftarrow C_u \cup R\\ \text{Return } K_R}$$

$$\frac{\textsc{EvSim}(u, \boldsymbol{x})}{\text{Require } \boldsymbol{x} \notin C_u\\ \text{If } b = 1\text{: } y \leftarrow \textsc{Ev}(u, \boldsymbol{x})\\ \text{If } b = 0\text{: } y \leftarrow\!\!\$\ f_u(\boldsymbol{x})\\ E_u \cup \{\boldsymbol{x}\}\\ \text{Return } y}$$

Let b^* denote the underlying chosen bit in $\mathsf{G}^{\mathsf{sim}^*\text{-ac-cprf}}$.

When b^* is 1. Con and Ev in the "hidden game" are returning actual constrained keys and evaluation results. Therefore, $\mathcal{A}_{\mathsf{cprf}^*}$ is indeed emulating the game $\mathsf{G}^{\mathsf{ind}^*\text{-ac-cprf}}$ for $\mathcal{A}_{\mathsf{cprf}}$.

When b^* is 0. First notice that E_u and C_u are consistent between the "emulated game" and the "hidden game". Therefore, when $b = 1$, Ev will always return a random value, just like the case in $b = 0$. Also, since $R \cap E_u = \emptyset$, simulator in the hidden game will not be told any information of prior queries. Thus, the output of Con is independent of possible Ev queries. For EvSim, in the "emulated game", we require $\boldsymbol{x} \notin C_u$, therefore, Ev in the "hidden game" will never satisfy the $\boldsymbol{x} \in C_u$ condition. Therefore, when b is 1, the output of Ev will always be the output of random function f_u. The requirement that $R \cap E_u = \emptyset$ in ConSim ensures that these output values are never shared with the simulator (and thus not used anywhere else). Thus there is no difference between these outputs and the outputs when b is 0. Therefore, $\mathcal{A}_{\mathsf{cprf}}$'s view is independent of b and its probability of guessing it correctly is exactly $1/2$. For any simulator S, we get

$$\begin{aligned}\mathsf{Adv}^{\mathsf{ind}^*\text{-ac-cprf}}_{\mathsf{CF},\mathsf{P},\mathcal{A}_{\mathsf{cprf}}}(\lambda) &= 2\Pr[b = b' | b^* = 1] - 1\\ &= 2(\mathsf{Adv}^{\mathsf{sim}^*\text{-ac-cprf}}_{\mathsf{CF},\mathsf{S},\mathsf{P},\mathcal{A}_{\mathsf{cprf}^*}}(\lambda) + \Pr[b = b' | b^* = 0]) - 1\\ &= 2\mathsf{Adv}^{\mathsf{sim}^*\text{-ac-cprf}}_{\mathsf{CF},\mathsf{S},\mathsf{P},\mathcal{A}_{\mathsf{cprf}^*}}(\lambda).\end{aligned}$$

By assumption there is a simulator that makes this negligible. □

B Searchable Symmetric Encryption Application

In this section, we show the details of application of our constrained PRF definition to searchable symmetric encryption (SSE) as summarized in Section 4.1. We first provide the syntax and security definitions we will use for randomized symmetric encryption. Then in Appendix B.1 we analyze SSE. We give detailed syntax and security definitions of a searchable symmetric scheme. Then we define the scheme FS2, which is a tweaked version of FS-RCPRF from [11]. We prove that it meets the corresponding security definition, when the underlying building blocks are secure.

Preliminaries: Symmetric Encryption. A symmetric encryption scheme SE specifies encryption algorithm SE.Enc, decryption algorithm SE.Dec, and key space SE.K. The syntax of its algorithms is given by $e \leftarrow\!\!\$\ \mathsf{SE.Enc}^{\mathsf{P}}(1^\lambda, ek, m)$ and $m \leftarrow \mathsf{SE.Dec}^{\mathsf{P}}(1^\lambda, ek, e)$ where $ek \in \mathsf{SE.K}(\lambda)$ is a key, m is a message to be encrypted, and e is a ciphertext. Informally, correctness requires that $\mathsf{SE.Dec}^{\mathsf{P}}(1^\lambda, ek, \mathsf{SE.Enc}^{\mathsf{P}}(1^\lambda, ek, m)) = m$.

SIM*-AC-CPA security [22,23] of SE is captured by the game shown in Fig. 6. In the "real world" ($b = 1$), the adversary can ask for arbitrary encryptions via Enc and ask to expose keys via Exp. In the "simulated world" ($b = 0$), encryption responses are chosen by a simulator which knows the length of the message and which user is doing the encryption. On exposure, the simulator is given all input-output pairs from prior Enc

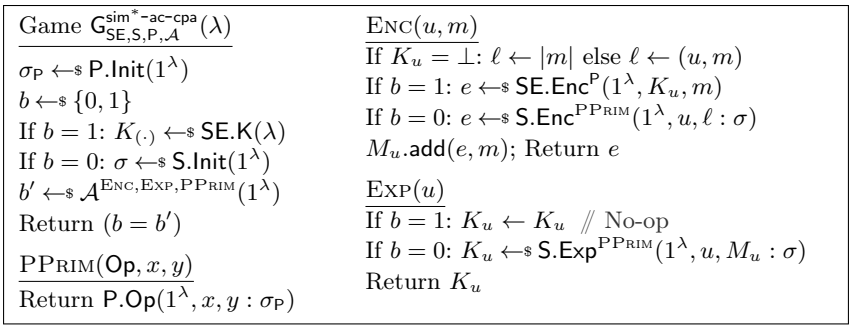


Fig. 6. Game defining the SIM*-AC CPA security of SE

queries to the given user, then outputs a key. Its goal is to reprogram P to make the previous input-output pairs look consistent with SE applied to this key.

The advantage of $\mathcal{A}$ is $\mathsf{Adv}^{\mathsf{sim^*\text{-}ac\text{-}cpa}}_{\mathsf{SE},\mathsf{S},\mathsf{P},\mathcal{A}}(\lambda) = 2\Pr[\mathsf{G}^{\mathsf{sim^*\text{-}ac\text{-}cpa}}_{\mathsf{SE},\mathsf{S},\mathsf{P},\mathcal{A}}(\lambda)] - 1$. If there exists a PPT S such that $\mathsf{Adv}^{\mathsf{sim^*\text{-}ac\text{-}cpa}}_{\mathsf{SE},\mathsf{S},\mathsf{P},\mathcal{A}}$ is negligible for all PPT $\mathcal{A}$, then we say that SE is SIM*-AC-CPA secure with P. Typical symmetric encryption schemes are known to be SIM*-AC-CPA secure [22,23].

B.1 Searchable Symmetric Encryption

Syntax. We consider a database DB represented as a set $\{(ind_i, W_i)\}_{i=1}^{|\mathsf{DB}|}$. Here each ind_i is the index of the i-th document and $W_i \subseteq \{0,1\}^*$ is the set of keywords corresponding to that document. A dynamic searchable encryption scheme SSE specifies algorithms SSE.Init, SSE.CSearch, SSE.SSearch, SSE.CUpdate, and SSE.SUpdate. We will only consider updates that add to the database (no deletion).

- $\mathsf{SSE.Init}(1^\lambda)$ outputs outputs a triple $(\mathsf{EDB}, K_{\mathsf{SSE}}, \mathbf{W})$ where K_{SSE} is a master secret key (stored by the client), EDB is an initial encrypted database (stored by the server), and $\mathbf{W}$ is a client state.[4]
- $(\mathsf{SSE.CSearch}(1^\lambda, K_{\mathsf{SSE}}, w : \mathbf{W}), \mathsf{SSE.SSearch}(1^\lambda, \tau : \mathsf{EDB}))$ form a one round search protocol between the client and server. The client's algorithm SSE.CSearch takes input master secret key K_{SSE}, keyword w, and state $\mathbf{W}$. It outputs a search token τ. The server's algorithm SSE.SSearch takes input search token τ and encrypted database EDB. It outputs is a set of document indices $I = \{ind_i\}$ corresponding to keyword w.
- $(\mathsf{SSE.CUpdate}(1^\lambda, K_{\mathsf{SSE}}, ind, w : \mathbf{W}), \mathsf{SSE.SUpdate}(1^\lambda, \tau : \mathsf{EDB}))$ forms an update protocol between the client and server. The client's algorithm SSE.CUpdate inputs K_{SSE}, index ind, keyword w, and $\mathbf{W}$. It outputs update token τ. The server's algorithm SSE.SUpdate takes in update token τ and encrypted database EDB.

Informally, correctness of an SSE scheme SSE requires that the search protocol returns set of document indices currently corresponding to the keyword w (with high probability).

Security. The security of a searchable symmetric encryption scheme guarantees that the server should not learn anything about the database and queries beyond some explicit leakage. We use a simulator-based, game-based proof to capture this idea. We define the game in Fig. 7. The adversary picks the database DB to start and is able to trigger any operations from the client via the four oracles Setup, Search, and Update, It is tasked to distinguish the real world ($b=1$) to the simulated world ($b=0$).

In the real world, the oracles will invoke the actual algorithms of SSE. In the simulated world, they will return output generated by a simulator S. The simulator is only provided with some leakage for each operation. This is captured by the *leakage function* L.

The advantage of $\mathcal{A}_{\mathsf{sse}}$ is defined as $\mathsf{Adv}^{\mathsf{sse}}_{\mathsf{SSE},\mathsf{L},\mathsf{S},\mathsf{P},\mathcal{A}_{\mathsf{sse}}}(\lambda) = 2\Pr[\mathsf{G}^{\mathsf{sse}}_{\mathsf{SSE},\mathsf{L},\mathsf{S},\mathsf{P},\mathcal{A}_{\mathsf{sse}}}(\lambda)] - 1$. If there exists a PPT S such that $\mathsf{Adv}^{\mathsf{sse}}_{\mathsf{SSE},\mathsf{L},\mathsf{S},\mathsf{P},\mathcal{A}_{\mathsf{sse}}}$ is negligible for all PPT $\mathcal{A}_{\mathsf{sse}}$, then we say that SSE is L secure with P. (Security is often defined to instead allow S to depend on $\mathcal{A}_{\mathsf{sse}}$, but constructions anyway achieve this stronger quantification.)

[4] Here our notation follows the analysis of FS-RCPRF by Bost, et al. [11] which assume an initally empty database.

Game $\mathsf{G}^{\mathsf{sse}}_{\mathsf{SSE},\mathsf{L},\mathsf{S},\mathsf{P},\mathcal{A}_{\mathsf{sse}}}(\lambda)$

$\sigma_{\mathsf{P}} \leftarrow\!\!\$\ \mathsf{P.Init}(1^\lambda)$
$b \leftarrow\!\!\$\ \{0,1\}$
If $b=1$: $(\mathsf{EDB}, K_{\mathsf{SSE}}, \mathbf{W}) \leftarrow\!\!\$\ \mathsf{SSE.Init}(1^\lambda)$
If $b=0$: $\sigma_{\mathsf{L}} \leftarrow\!\!\$\ \mathsf{L.Init}(1^\lambda)$
$(\mathsf{EDB}, \sigma) \leftarrow\!\!\$\ \mathsf{S.Init}^{\mathrm{PP}_{\mathrm{RIM}}}(1^\lambda)$
$b' \leftarrow\!\!\$\ \mathcal{A}^{\mathrm{Search},\mathrm{Update},\mathrm{PP}_{\mathrm{RIM}}}_{\mathsf{sse}}(1^\lambda, \mathsf{EDB})$
Return $(b = b')$

$\mathrm{PP}_{\mathrm{RIM}}(\mathsf{Op}, x, y)$
Return $\mathsf{P.Op}(1^\lambda, x, y : \sigma_{\mathsf{P}})$

$\mathrm{Search}(w)$
If $b=1$: $\tau \leftarrow\!\!\$\ \mathsf{SSE.CSearch}^{\mathsf{P}}(1^\lambda, K_{\mathsf{SSE}}, w : \mathbf{W})$
If $b=0$: $\ell \leftarrow\!\!\$\ \mathsf{L.CSearch}(1^\lambda, w : \sigma_{\mathsf{L}})$
$\tau \leftarrow\!\!\$\ \mathsf{S.CSearch}^{\mathrm{PP}_{\mathrm{RIM}}}(1^\lambda, \ell : \sigma)$
Return τ

$\mathrm{Update}(ind, w)$
If $b=1$: $\tau \leftarrow\!\!\$\ \mathsf{SSE.CUpdate}^{\mathsf{P}}(1^\lambda, K_{\mathsf{SSE}}, ind, w : \mathbf{W})$
If $b=0$: $\ell \leftarrow\!\!\$\ \mathsf{L.CUpdate}(1^\lambda, ind, w : \sigma_{\mathsf{L}})$
$\tau \leftarrow\!\!\$\ \mathsf{S.CUpdate}^{\mathrm{PP}_{\mathrm{RIM}}}(1^\lambda, \ell : \sigma)$
Return τ

Fig. 7. Game defining security of searchable symmetric encryption scheme SSE with respect to leakage L

$\mathsf{FS2.Init}(1^\lambda)$
$K_{\mathsf{SSE}} \leftarrow\!\!\$\ \mathsf{F.K}(\lambda)$
$\mathbf{W}[\cdot] \leftarrow 0$; $\mathsf{EDB} \leftarrow [\cdot]$
Return $(\mathsf{EDB}, K_{\mathsf{SSE}}, \mathbf{W})$

$\mathsf{FS2.CUpdate}^{\mathsf{P}}(1^\lambda, K_{\mathsf{SSE}}, w, ind : \mathbf{W})$
$K_w \leftarrow \mathsf{F.Ev}^{\mathsf{P}}(1^\lambda, K_{\mathsf{SSE}}, w)$
$c \leftarrow \mathbf{W}[w]$; $\mathbf{W}[w] \leftarrow c+1$
$(UT, ek) \leftarrow \mathsf{CF.Ev}^{\mathsf{P}}(1^\lambda, K_w, 1^{c+1})$
$e \leftarrow\!\!\$\ \mathsf{SE.Enc}^{\mathsf{P}}(1^\lambda, ek, ind)$
$\tau \leftarrow (UT, e)$
Return τ

$\mathsf{FS2.SUpdate}^{\mathsf{P}}(1^\lambda, \tau = (UT, e) : \mathsf{EDB})$
$\mathsf{EDB}[UT] \leftarrow e$
Return

$\mathsf{FS2.CSearch}^{\mathsf{P}}(1^\lambda, K_{\mathsf{SSE}}, w : \mathbf{W})$
$c \leftarrow \mathbf{W}[w]$
If $c=0$: Return $(\perp, 0)$
$K_w \leftarrow \mathsf{F.Ev}^{\mathsf{P}}(1^\lambda, K_{\mathsf{SSE}}, w)$
$k_{[c]} \leftarrow \mathsf{CF.Con}^{\mathsf{P}}(1^\lambda, K_w, [c])$
$\tau \leftarrow (k_{[c]}, c)$
Return τ

$\mathsf{FS2.SSearch}^{\mathsf{P}}(1^\lambda, \tau = (k_{[c]}, c) : \mathsf{EDB})$
$D \leftarrow \emptyset$
If $c=0$: Return D
For $i = c, \ldots, 1$:
  $(UT, ek) \leftarrow \mathsf{CF.CEv}^{\mathsf{P}}(1^\lambda, k_{[c]}, [c], 1^i)$
  $e \leftarrow \mathsf{EDB}[UT]$
  $ind_i \leftarrow \mathsf{SE.Dec}^{\mathsf{P}}(1^\lambda, ek, e)$
  $D.\mathsf{add}(ind_i)$
Return D

Fig. 8. Algorithms of forward-secure searchable encryption scheme FS2

Construction. Let SE be a symmetric encryption scheme. Let CF be a constrained function family with $\mathsf{CF.Inp} = 1^*$, $\mathsf{CF.Out} = \mathsf{CF.Out}_1 \times \mathsf{SE.K}$, and CF.SS containing $[c] = \{1, 11, \ldots, 1^c\}$ for all $c \in \mathbb{N}$.[5] Let F be a function family with $\mathsf{F.Out} = \mathsf{CF.K}$. Then FS2 is defined as shown in Fig. 8. The idea behind this construction was described in Section 4.1.

Leakage. On update queries, FS2 does not leak anything beyond the length of ind because the output of CF on a new input will look random, so UT looks random and e is a secure encryption. On searches, the adversary can match the constrained key it sees with earlier updates and searches. It can learns the document indices by performing the actual search procedure (and when the keyword was added to that particular document by comparing with earlier indices). Let t be a timestamp that starts at 0 and increases with each query. For simplicity, we write $\mathsf{Q.add}(w)$, which means setting $\mathsf{Q}[t] \leftarrow w$ where t is the current timestamp. $\mathsf{U.add}(ind, w)$ should be interpreted similarly. This is formalized by the leakage function $\mathsf{L_{fs}}$ shown below.

[5] It suffices to restrict CF.Inp to inputs of length at most 2^λ. Such a CF can be constructed from a prefix-constrained function family CF′ with domain $\mathsf{CF'.Inp}(\lambda) = \{0,1\}^\lambda$, by interpreting elements of CF.Inp and CF′.Inp as integers (written in unary and binary respectively).

$\mathsf{L_{fs}.Init}(1^\lambda)$	$\mathsf{L_{fs}.CUpdate}(1^\lambda, ind, w : (\mathsf{Q}, \mathsf{U}))$	$\mathsf{L_{fs}.CSearch}(1^\lambda, w : (\mathsf{Q}, \mathsf{U}))$
$\sigma_\mathsf{L} \leftarrow (\emptyset, \emptyset)$	$\mathsf{U.add}(ind, w)$	$\mathsf{Q.add}(w)$
Return σ_L	Return $\lvert ind \rvert$	$\mathsf{sp} \leftarrow \{t \ : \ \mathsf{Q}[t] = w\}$
		$\mathsf{UpHist} \leftarrow \{(t, ind) \ : \ \mathsf{U}[t] = (ind, w)\}$
		Return $(\mathsf{sp}, \mathsf{UpHist})$

Because updates leak nothing unless the keyword is later searched, FS2 can be considered forward secure. We do not actually require the search pattern sp in our simulation. When the update history UpHist for w is non-empty, it implicitly allows us to be consistent between different searches for w. When UpHist is empty we just return $\tau = (\perp, 0)$.

Security Result. The following theorem captures that security of FS2 follows from the security of its underlying components.

Theorem 6. *Suppose* F *is IND*-PRF secure with* P*,* CF *is SIM*-AC-PRF secure with* P*, and* SE *is SIM*-AC-CPA secure with* P*. Then* FS2 *is* $\mathsf{L_{FS}}$ *secure with* P*.*

A similar proof [23] analyzing the SSE scheme Π_{bas} needed to assume a "key-private" security notion for the underlying symmetric encryption, which we do not have to do here. The difference is that for that scheme every ciphertext for a given keyword was encrypted with the same key while in FS2 each ciphertext comes from a different key.

Proof. The proof proceeds via a series of hybrids G_0 to G_3, switching from the real world to the simulated world. In these hybrids we first switch the output of F to be random, then switch the output of CF to be simulated, then switch the output of SE to be simulated. We define a simulator S from the simulators $\mathsf{S_{cprf}}$ for CF and $\mathsf{S_{cpa}}$ for SE and build adversaries $\mathcal{A}_{\mathsf{prf}}$, $\mathcal{A}_{\mathsf{cprf}^*}$, and $\mathcal{A}_{\mathsf{cpa}}$ for which we establish the following claims:

1. $\Pr[\mathsf{G}_0(\lambda)] = \Pr[\mathsf{G}^{\mathsf{sse}}_{\mathsf{SSE},\mathsf{L_{FS}},\mathsf{S},\mathsf{P},\mathcal{A}_{\mathsf{sse}}}(\lambda) | b = 1]$
2. $\Pr[\mathsf{G}_0(\lambda)] - \Pr[\mathsf{G}_1(\lambda)] = \mathsf{Adv}^{\mathsf{ind^*\text{-}prf}}_{\mathsf{F},\mathsf{P},\mathcal{A}_{\mathsf{prf}}}(\lambda)$
3. $\Pr[\mathsf{G}_1(\lambda)] - \Pr[\mathsf{G}_2(\lambda)] = \mathsf{Adv}^{\mathsf{sim^*\text{-}ac\text{-}cprf}}_{\mathsf{CF},\mathsf{S_{cprf}},\mathsf{P},\mathcal{A}_{\mathsf{cprf}^*}}(\lambda)$
4. $\Pr[\mathsf{G}_2(\lambda)] - \Pr[\mathsf{G}_3(\lambda)] = \mathsf{Adv}^{\mathsf{sim^*\text{-}ac\text{-}cpa}}_{\mathsf{SE},\mathsf{S_{cpa}},\mathsf{P},\mathcal{A}_{\mathsf{cpa}}}(\lambda)$
5. $\Pr[\mathsf{G}_3(\lambda)] = \Pr[\neg \mathsf{G}^{\mathsf{sse}}_{\mathsf{SSE},\mathsf{L_{FS}},\mathsf{S},\mathsf{P},\mathcal{A}_{\mathsf{sse}}}(\lambda) | b = 0]$.

Therefore,

$$\begin{aligned}\mathsf{Adv}^{\mathsf{sse}}_{\mathsf{SSE},\mathsf{L_{FS}},\mathsf{S},\mathsf{P},\mathcal{A}_{\mathsf{sse}}}(\lambda) &= \Pr[\mathsf{G}_0(\lambda)] - \Pr[\mathsf{G}_3(\lambda)] = \sum_{i=0}^{2} \Pr[\mathsf{G}_i(\lambda)] - \Pr[\mathsf{G}_{i+1}(\lambda)] \\ &= \mathsf{Adv}^{\mathsf{ind^*\text{-}prf}}_{\mathsf{F},\mathsf{P},\mathcal{A}_{\mathsf{prf}}}(\lambda) + \mathsf{Adv}^{\mathsf{sim^*\text{-}ac\text{-}cprf}}_{\mathsf{CF},\mathsf{S_{cprf}},\mathsf{P},\mathcal{A}_{\mathsf{cprf}^*}}(\lambda) + \mathsf{Adv}^{\mathsf{sim^*\text{-}ac\text{-}cpa}}_{\mathsf{SE},\mathsf{S_{cpa}},\mathsf{P},\mathcal{A}_{\mathsf{cpa}}}(\lambda).\end{aligned}$$

The algorithms we construct are PPT (assuming the given algorithms are), so this implies the result by letting $\mathsf{S_{cprf}}$ and $\mathsf{S_{cpa}}$ be the simulators assumed to exist by the security of CF and SE.

Simulator. The simulator S we will use for FS2 is given in Fig. 9. It responds to update queries by choosing UT at random and having $\mathsf{S_{cpa}}$ choose the ciphertext e. It tracks these in the list H. Then for search queries it wants to use $\mathsf{S_{cprf}}$ to choose the constrained key. For this it needs to give $\mathsf{S_{cprf}}$ a partial function telling it what the "outputs" of CF from earlier updates. It builds this in T using a for loop where it remembers the UT's from earlier updates and asks $\mathsf{S_{cpa}}$ to decide what the ek's should have been.

For asking the simulators to expose keys, we need to tell them the "name" of the corresponding user. We name the $\mathsf{S_{cpa}}$ users by the "time" at which their ciphertext was output by an update and $\mathsf{S_{cprf}}$ users by the first "time" the corresponding keyword was included in an update.

$\underline{\mathsf{S.Init}(1^\lambda)}$
$\mathsf{EDB} \leftarrow [\cdot]$
$\sigma_{\mathsf{cprf}} \leftarrow\!\!\$\ \mathsf{S}_{\mathsf{cprf}}.\mathsf{Init}(1^\lambda)$
$\sigma_{\mathsf{cpa}} \leftarrow\!\!\$\ \mathsf{S}_{\mathsf{cpa}}.\mathsf{Init}(1^\lambda)$
$\mathsf{H} \leftarrow \emptyset$
$\mathbf{V} \leftarrow 0$
Return $(\mathsf{EDB}, (\sigma_{\mathsf{cprf}}, \sigma_{\mathsf{cpa}}, \mathsf{H}))$

$\underline{\mathsf{S.CUpdate}^{\mathrm{PP_{RIM}}}(1^\lambda, \ell : (\sigma_{\mathsf{cprf}}, \sigma_{\mathsf{cpa}}, \mathsf{H}, \mathbf{V}))}$
$UT \leftarrow\!\!\$\ \mathsf{CF.Out}_1(\lambda)$
$t \leftarrow |\mathsf{H}| + 1$
$e \leftarrow\!\!\$\ \mathsf{S}_{\mathsf{cpa}}.\mathsf{Enc}^{\mathrm{PP_{RIM}}}(1^\lambda, t, \ell : \sigma_{\mathsf{cpa}})$
$\mathsf{H.add}(UT, e)$
Return (UT, e)

$\underline{\mathsf{S.CSearch}^{\mathrm{PP_{RIM}}}(1^\lambda, (\mathsf{sp}, \mathsf{UpHist}) : (\sigma_{\mathsf{cprf}}, \sigma_{\mathsf{cpa}}, \mathsf{H}, \mathbf{V}))}$
$\{(t_1, ind_1), (t_2, ind_2), \ldots, (t_c, ind_c)\} \leftarrow \mathsf{UpHist}$
// $t_i < t_{i+1}$
If $c = 0$: Return $(\perp, 0)$
$c_{prev} \leftarrow \mathbf{V}[t_1]$; $\mathbf{V}[t_1] \leftarrow c$
$T \leftarrow [\cdot]$
For $i = c_{prev} + 1, \ldots, c$:
  $(UT, e) \leftarrow \mathsf{H}[t_i]$
  $M \leftarrow \emptyset$; $M.\mathsf{add}(e, ind_i)$
  $ek \leftarrow\!\!\$\ \mathsf{S}_{\mathsf{cpa}}.\mathsf{Exp}^{\mathrm{PP_{RIM}}}(1^\lambda, t_i, M : \sigma_{\mathsf{cpa}})$
  $T[i] \leftarrow (UT, ek)$
$k_{[c]} \leftarrow\!\!\$\ \mathsf{S}_{\mathsf{cprf}}.\mathsf{Con}^{\mathrm{PP_{RIM}}}(1^\lambda, t_1, [c], T : \sigma_{\mathsf{cprf}})$
Return $(k_{[c]}, c)$

Fig. 9. Simulator S for SSE scheme $\mathsf{FS2}$, constructed from simulator $\mathsf{S}_{\mathsf{cprf}}$ for CF and simulator $\mathsf{S}_{\mathsf{cpa}}$ for SE

Games $\mathsf{G}_0(\lambda)$, $\boxed{\mathsf{G}_1(\lambda)}$
$\sigma_{\mathsf{P}} \leftarrow\!\!\$\ \mathsf{P.Init}(1^\lambda)$
$K_{\mathsf{SSE}} \leftarrow\!\!\$\ \mathsf{F.K}(\lambda)$
$\boxed{K_{(\cdot)} \leftarrow\!\!\$\ \mathsf{CF.K}(\lambda)}$
$\mathbf{W}[\cdot] \leftarrow 0$
$b' \leftarrow\!\!\$\ \mathcal{A}_{\mathsf{sse}}^{\textsc{Search},\textsc{Update},\mathrm{PP_{RIM}}}(1^\lambda, [\cdot])$
Return $(b' = 1)$

$\underline{\textsc{PPrim}(\mathsf{Op}, x, y)}$
Return $\mathsf{P.Op}(x, y : \sigma_{\mathsf{P}})$

$\underline{\textsc{Search}(w)}$
$c \leftarrow \mathbf{W}[w]$
If $c = 0$: Return $(\perp, 0)$
$K_w \leftarrow \mathsf{F.Ev}^{\mathsf{P}}(1^\lambda, K_{\mathsf{SSE}}, w)$
$k_{[c]} \leftarrow \mathsf{CF.Con}^{\mathsf{P}}(K_w, [c])$
Return $(k_{[c]}, c)$

$\underline{\textsc{Update}(ind, w)}$
$K_w \leftarrow \mathsf{F.Ev}^{\mathsf{P}}(1^\lambda, K_{\mathsf{SSE}}, w)$
$c \leftarrow \mathbf{W}[w]$; $\mathbf{W}[w] \leftarrow c + 1$
$(UT, ek) \leftarrow \mathsf{CF.Ev}^{\mathsf{P}}(1^\lambda, K_w, 1^{c+1})$
$e \leftarrow\!\!\$\ \mathsf{SE.Enc}^{\mathsf{P}}(1^\lambda, ek, ind)$
Return (UT, e)

Fig. 10. Hybrid games G_0 and G_1 used in analysis of $\mathsf{FS2}$

Adversary $\mathcal{A}_{\mathsf{prf}}^{\textsc{Ev},\mathrm{PP_{RIM}}}(1^\lambda)$
$\mathbf{W}[\cdot] \leftarrow 0$
$b' \leftarrow\!\!\$\ \mathcal{A}_{\mathsf{sse}}^{\textsc{Search},\textsc{Update},\mathrm{PP_{RIM}}}(1^\lambda, [\cdot])$
Return b'

$\underline{\textsc{Search}(w)}$
$c \leftarrow \mathbf{W}[w]$
If $c = 0$: Return $(\perp, 0)$
$K_w \leftarrow \textsc{Ev}(w)$
$k_{[c]} \leftarrow \mathsf{CF.Con}^{\mathsf{P}}(K_w, [c])$
Return $(k_{[c]}, c)$

$\underline{\textsc{Update}(ind, w)}$
$K_w \leftarrow \textsc{Ev}(w)$
$c \leftarrow \mathbf{W}[w]$; $\mathbf{W}[w] \leftarrow c + 1$
$(UT, ek) \leftarrow \mathsf{CF.Ev}^{\mathsf{P}}(1^\lambda, K_w, 1^{c+1})$
$e \leftarrow\!\!\$\ \mathsf{SE.Enc}^{\mathsf{P}}(1^\lambda, ek, ind)$
Return (UT, e)

Fig. 11. IND*-PRF adversary $\mathcal{A}_{\mathsf{prf}}$ used to switch the output of F from real to random

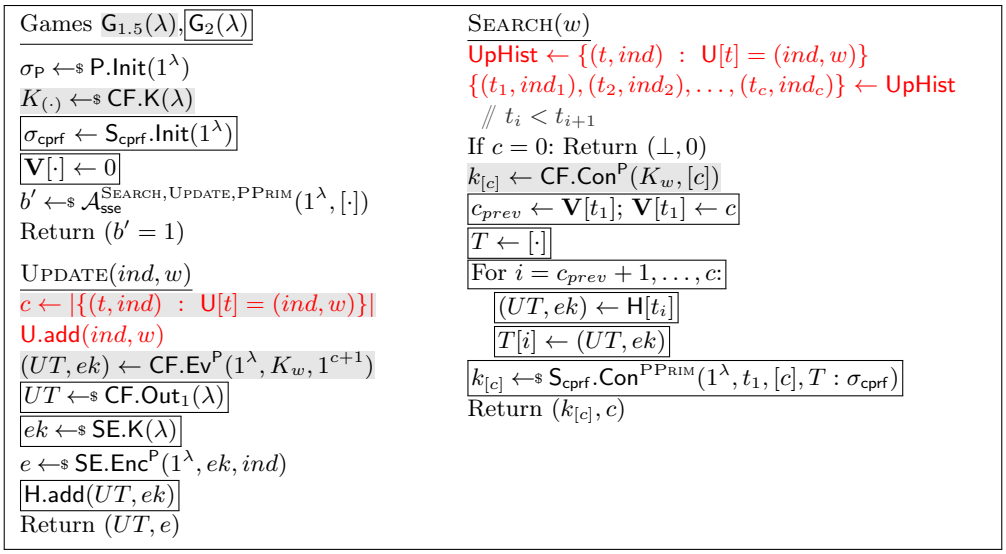


Fig. 12. Hybrids game $\mathsf{G}_{1.5}$ and G_2 used in analysis of FS2

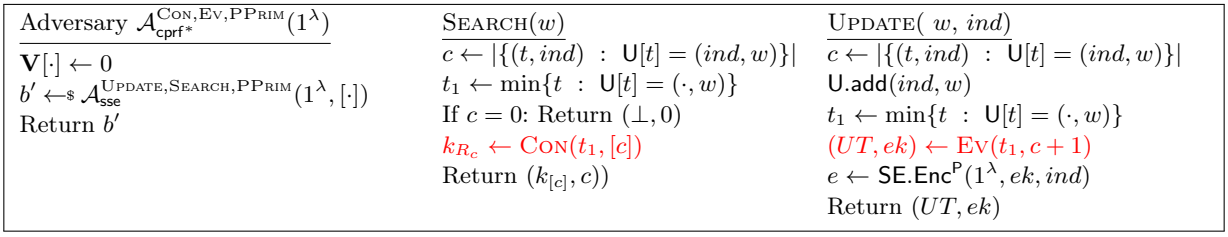


Fig. 13. SIM*-AC-CPRF adversary $\mathcal{A}_{\mathsf{cprf}^*}$ used to switch the output of CF from real to simulated

Claims 1 and 2. We start with G_0 defined in Fig. 11 which contains highlighted but not boxed code. It was obtained by plugging in the code of SSE.Init, SSE.CSearch, and SSE.CUpdate into the $b = 1$ world of $\mathsf{G}^{\mathsf{sse}}_{\mathsf{SSE},\mathsf{L}_{\mathsf{fs}},\mathsf{S},\mathsf{P},\mathcal{A}_{\mathsf{sse}}}(\lambda)$. Hence our first claim, that $\Pr[\mathsf{G}_0(\lambda)] = \Pr[\mathsf{G}^{\mathsf{sse}}_{\mathsf{SSE},\mathsf{L}_{\mathsf{FS}},\mathsf{S},\mathsf{P},\mathcal{A}_{\mathsf{sse}}}(\lambda) | b = 1]$, follows.

Game G_1 in Fig. 11 contains boxed but not highlighted code. We replace the usage of F by picking $K_{(\cdot)}$ randomly. Naturally, we will bound the difference between G_0 and G_1 by the IND*-PRF security of F. The relevant adversary $\mathcal{A}_{\mathsf{prf}}$ is given in Fig. 10. It simulates the view of $\mathcal{A}_{\mathsf{sse}}$ in these two games by using its Ev oracle to compute K_w. Correctness of this simulation gives our second claim, that $\Pr[\mathsf{G}_0(\lambda)] - \Pr[\mathsf{G}_1(\lambda)] = \mathsf{Adv}^{\mathsf{ind}^*\text{-}\mathsf{prf}}_{\mathsf{F},\mathsf{P},\mathcal{A}_{\mathsf{prf}}}(\lambda)$.

Our games never modify PPrim, so henceforth we will omit its code.

Claim 3. Now consider the games defined in Fig. 12. The first game $\mathsf{G}_{1.5}$ contains highlighted but not boxed code. It is identical to G_1 except that we have updated the booking to omit the table $\mathbf{W}$. These changes are colored in red. We use a list U to store all of the Update queries made (matching the list stored by L) and when $c = \mathbf{W}[w]$ is needed, we recover it by checking how many entries of U contain w. Hence $\Pr[\mathsf{G}_{1.5}(\lambda)] = \Pr[\mathsf{G}_1(\lambda)]$.

Now game G_2 contains boxed but not highlighted code. It switches output of CF to being simulated. In Update we sample (UT, ek) at random and store this in a list H (matching the list stored by S, except it contains ek rather than e). In Search we have $\mathsf{S}_{\mathsf{cprf}}$ choose the constrained key. To do so, we need to give it a partial function containing all the random values output by Update for this keyword since the last time a Search query was made for this keyword. We use list H to fill the partial function T, with table $\mathbf{V}$ tracking the value of c from the previous Search query for this keyword to know what values of the list to

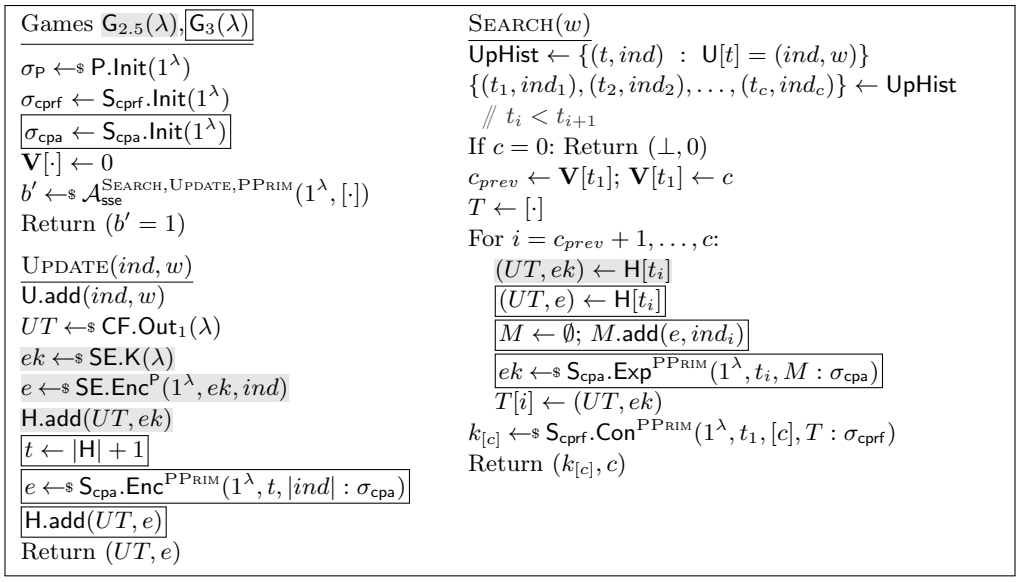


Fig. 14. Hybrid games $\mathsf{G}_{2.5}$ and G_3 used in analysis of FS2

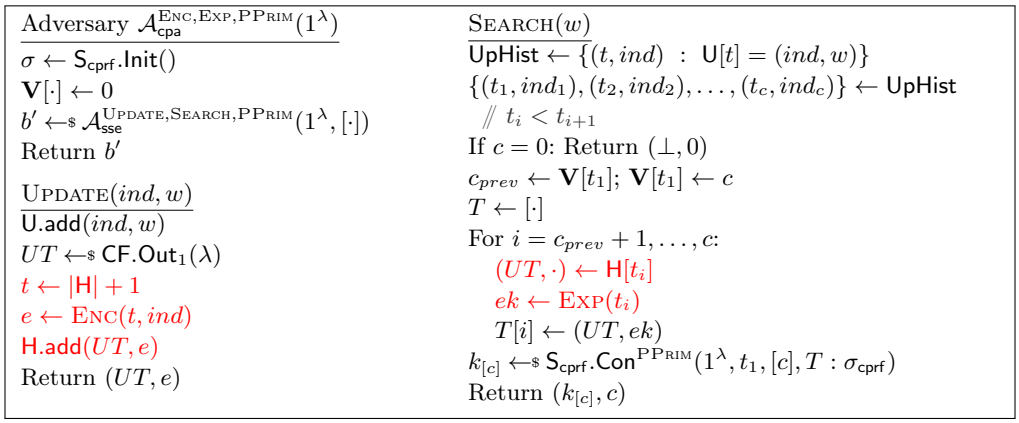


Fig. 15. SIM*-AC-CPA adversary $\mathcal{A}_{\mathsf{cpa}}$ is used to switch the output of SE from real to simulated

use. Following S, we name the user for a keyword (from $\mathsf{S}_{\mathsf{cprf}}$'s perspective) after the first time that keyword was updated.

We can reduce the difference between G_1 and G_2 to SIM*-AC-CPRF security of CF using the adversary $\mathcal{A}_{\mathsf{cprf}*}$ in Fig. 13. It simulates the view of $\mathcal{A}_{\mathsf{sse}}$ in these two games by using its oracles to choose $k_{[c]}$, UT, and ek. Correctness of this simulation gives our third claim, that $\Pr[\mathsf{G}_1(\lambda)] - \Pr[\mathsf{G}_2(\lambda)] = \mathsf{Adv}^{\mathsf{sim}^*\text{-ac-cprf}}_{\mathsf{CF},\mathsf{S}_{\mathsf{cprf}},\mathsf{P},\mathcal{A}_{\mathsf{cprf}*}}(\lambda)$.

Claim 4. Now consider the games defined in Fig. 14. Game $\mathsf{G}_{2.5}$ contains highlighted but not boxed code. It was copied directly from G_2. Hence $\Pr[\mathsf{G}_{2.5}(\lambda)] = \Pr[\mathsf{G}_2(\lambda)]$.

Now game G_3 contains boxed but not highlighted code. It switches from using real encryptions and keys of SE to using simulated ones. In Update we follow S and use the time of the current query as the name of the user for $\mathsf{S}_{\mathsf{cpa}}$. We switch to storing e, rather than ek in H. In Search we need ek to give to $\mathsf{S}_{\mathsf{cprf}}$, so we recover it using $\mathsf{S}_{\mathsf{cpa}}$. We tell $\mathsf{S}_{\mathsf{cpa}}$ the appropriate document index ind from U.

We can reduce the difference between G_2 and G_3 to the SIM*-AC-CPA security of SE using the adversary $\mathcal{A}_{\mathsf{cpa}}$ in Fig. 15. It simulated the view of $\mathcal{A}_{\mathsf{sse}}$ in these two games by using its own oracles to choose e and ek. Correctness of this simulation gives our fourth claim, that $\Pr[\mathsf{G}_2(\lambda)] - \Pr[\mathsf{G}_3(\lambda)] = \mathsf{Adv}^{\mathsf{sim^*\text{-}ac\text{-}cpa}}_{\mathsf{SE},\mathsf{S}_{\mathsf{cpa}},\mathsf{P},\mathcal{A}_{\mathsf{cpa}}}(\lambda)$.

Claim 5. Game G_3 responds to oracle queries identically to the $b = 0$ world of $\mathsf{G}^{\mathsf{sse}}$. Plugging the code of L_{FS} and S into the latter can verify this claim. Consequently, their only difference is that the former outputs the boolean ($b' = 1$) while the latter outputs the boolean ($b' = 0$). This gives our fifth claim, that $\Pr[\mathsf{G}_3(\lambda)] = \Pr[\neg \mathsf{G}^{\mathsf{sse}}_{\mathsf{SSE},\mathsf{L}_{\mathsf{FS}},\mathsf{S},\mathsf{P},\mathcal{A}_{\mathsf{sse}}}(\lambda) | b = 0]$. □

C Puncturable Key Wrapping Application

In this section, we show the details of our application of our constrained PRF definition to puncturable key wrapping as summarized in Section 4.2. We first provide the syntax and security definitions that we will use for nonce-based symmetric encryption (in particular providing a misuse-resistant SIM*-AC notion). Then in Appendix C.1 we give syntax and security notions of PKW as formalized by Backendal, Günther, and Paterson [2]. We consider the batch puncturing setting discussed, but not analyzed, by Backendal, et al. We confirm that their construction from a constrained PRF and encryption scheme is secure in this setting assuming SIM*-AC security of both components.

Preliminaries: Nonce-based Encryption. A nonce-based encryption scheme NE specifies encryption algorithm $\mathsf{NE.Enc}$ and decryption algorithm $\mathsf{NE.Dec}$ together with key space $\mathsf{NE.K}$, nonce space $\mathsf{NE.N}$, header space $\mathsf{NE.H}$, message space $\mathsf{NE.M}$, and ciphertext length function $\mathsf{NE.cl}$. The syntax of its algorithms are $c \leftarrow \mathsf{NE.Enc}^{\mathsf{P}}(1^\lambda, sk, n, h, m)$ and $m \leftarrow \mathsf{NE.Dec}^{\mathsf{P}}(1^\lambda, sk, n, h, c)$ where $sk \in \mathsf{NE.K}(\lambda)$ is the key, $n \in \mathsf{NE.N}(\lambda)$ is the nonce, $h \in \mathsf{NE.H}(\lambda)$ is the header, $m \in \mathsf{NE.M}(\lambda)$ is the message, and $c \in \{0,1\}^{\mathsf{NE.cl}(\lambda,|m|)}$ is the ciphertext. Informally, correctness requires that $m = \mathsf{NE.Dec}^{\mathsf{P}}(1^\lambda, sk, n, h, \mathsf{NE.Enc}^{\mathsf{P}}(1^\lambda, sk, n, h, m))$ always holds.

We define a SIM*-AC-style security notion for NE which captures misuse resistance [28] and that ciphertexts should be indistinguishable from random. It is captured by the game shown in Fig. 16. In the "real world" ($b = 1$), the adversary can ask for arbitrary encryptions via Enc and ask to expose keys via Exp. In the "simulated world" ($b = 0$), encryption responses are sampled at random and kept consistent across repeat queries. On exposure, the simulator is given all input-output pairs for prior Enc queries to the given user, then outputs a key. Its goal is to reprogram P to make the previous input-output pairs look consistent with NE applied to this key. Misuse resistance is captured because no restrictions are placed on nonces repeating.

The advantage of $\mathcal{A}$ is $\mathsf{Adv}^{\mathsf{sim^*\text{-}ac\text{-}mr\$}}_{\mathsf{NE},\mathsf{S},\mathsf{P},\mathcal{A}}(\lambda) = 2\Pr[\mathsf{G}^{\mathsf{sim^*\text{-}ac\text{-}mr\$}}_{\mathsf{NE},\mathsf{S},\mathsf{P},\mathcal{A}}(\lambda)] - 1$. If there exists a PPT S such that $\mathsf{Adv}^{\mathsf{sim^*\text{-}ac\text{-}mr\$}}_{\mathsf{NE},\mathsf{S},\mathsf{P},\mathcal{A}}$ is negligible for all PPT $\mathcal{A}$, then we say NE is SIM*-AC-MR$ secure with P. We expect that "typical" misuse-resistant encryption schemes achieve SIM*-AC-MR$ security assuming appropriate security of their building blocks.

C.1 Puncturable Key Wrapping

Syntax. A puncturable key wrapping scheme [2] specifies key generation algorithm $\mathsf{PKW.Kg}$, wrapping algorithm $\mathsf{PKW.Wrap}$, unwrapping algorithm $\mathsf{PKW.Unwrap}$, and puncturing algorithm $\mathsf{PKW.Punc}$ together with tag space $\mathsf{PKW.T}$, header space $\mathsf{PKW.H}$, message space $\mathsf{PKW.M}$, and ciphertext length function $\mathsf{PKW.cl}$. The syntax of its algorithms is given by $K \leftarrow\!\!\$\ \mathsf{PKW.Kg}(1^\lambda)$, $c \leftarrow\!\!\$\ \mathsf{PKW.Wrap}^{\mathsf{P}}(1^\lambda, K, t, h, m)$, $m \leftarrow \mathsf{PKW.Unwrap}^{\mathsf{P}}(1^\lambda, K, t, h, c)$, and $K' \leftarrow \mathsf{PKW.Punc}^{\mathsf{P}}(1^\lambda, K, t)$ where K is a key, K' is a key punctured at t, $t \in \mathsf{PKW.T}(\lambda)$ is a tag, $h \in \mathsf{PKW.H}(\lambda)$ is a header, $m \in \mathsf{PKW.M}(\lambda)$ is a message, and $c \in \{0,1\}^{\mathsf{PKW.cl}(\lambda,|m|)}$ is a ciphertext. The expectation is that K' "works equivalently" to K except on the tag t — in particular, wrapping or unwrapping with K' and t should return $\perp$. Informally, correctness of PKW requires that $m = \mathsf{PKW.Unwrap}^{\mathsf{P}}(1^\lambda, K'', t, h, \mathsf{PKW.Wrap}^{\mathsf{P}}(1^\lambda, K', t, h, m))$ if K', K'' were derived from some initial key $K \leftarrow\!\!\$\ \mathsf{PKW.Kg}(1^\lambda)$ by puncturing on tags other than t.

In essence, a puncturable key wrapping scheme is a nonce-based encryption scheme with added functionality for puncturing the key. The differing terminology (wrapping, unwrapping, tags vs. encrypting, decrypting, nonces) hints at the primitive being intended for use with messages that are themselves keys.

Game $\mathsf{G}^{\mathsf{sim}^*\text{-ac-mr\$}}_{\mathsf{NE},\mathsf{S},\mathsf{P},\mathcal{A}}(\lambda)$

$\sigma_{\mathsf{P}} \leftarrow\$ \mathsf{P.Init}(1^\lambda)$
$b \leftarrow\$ \{0,1\}$
If $b = 1$: $K_{(\cdot)} \leftarrow\$ \mathsf{SE.K}(\lambda)$
If $b = 0$: $\sigma \leftarrow\$ \mathsf{S.Init}(1^\lambda)$
$b' \leftarrow\$ \mathcal{A}^{\textsc{Enc},\textsc{Exp},\textsc{PPrim}}(1^\lambda)$
Return $(b = b')$

$\textsc{PPrim}(\mathsf{Op}, x, y)$
$y \leftarrow\$ \mathsf{P.Op}(1^\lambda, x, y : \sigma_{\mathsf{P}})$
Return y

$\textsc{Enc}(u, n, h, m)$
If $b = 1$: $T_u[n,h,m] \leftarrow \mathsf{NE.Enc}^{\mathsf{P}}(1^\lambda, K_u, n, h, m)$
If $b = 0$:
  If $K_u \neq \bot$: $T_u[n,h,m] \leftarrow\$ \mathsf{S.Enc}^{\mathrm{PP}_{\textsc{Prim}}}(1^\lambda, u, n, h, m : \sigma)$
  Else if $T_u[n,h,m] \neq \bot$: $T_u[n,h,m] \leftarrow T_u[n,h,m]$ // No-op
  Else $T_u[n,h,m] \leftarrow\$ \{0,1\}^{\mathsf{NE.cl}(\lambda,|m|)}$
Return $T_u[n,h,m]$

$\textsc{Exp}(u)$
If $b = 1$: $K_u \leftarrow K_u$ // No-op
If $b = 0$: $K_u \leftarrow\$ \mathsf{S.Exp}^{\mathrm{PP}_{\textsc{Prim}}}(1^\lambda, u, T_u : \sigma)$
Return K_u

Fig. 16. Game defining SIM*-AC-MR$ security of NE

Game $\mathsf{G}^{\mathsf{find\$^*\text{-rcpa}}}_{\mathsf{PKW},\mathsf{P},\mathcal{A}}(\lambda)$, $\boxed{\mathsf{G}^{\mathsf{find\$\text{-rcpa+}}}_{\mathsf{PKW},\mathsf{P},\mathcal{A}}(\lambda)}$

$\sigma_{\mathsf{P}} \leftarrow\$ \mathsf{P.Init}(1^\lambda)$
$K_{(\cdot)} \leftarrow\$ \mathsf{PKW.Kg}(1^\lambda)$
$b \leftarrow\$ \{0,1\}$
$b' \leftarrow\$ \mathcal{A}^{\text{RO\$-}\textsc{Wrap},\textsc{Wrap},\textsc{Punc},\textsc{Corrupt},\textsc{PPrim}}(1^\lambda)$
Return $(b = b')$

RO\$-$\textsc{Wrap}(u, t, h, m)$
Require $t \notin W_u$
$\boxed{\text{Require } (t,h,m) \notin W_u}$
Require $t \notin P_u$ and $\mathsf{corr}_u = \mathsf{false}$
If $b = 1$: $c \leftarrow \mathsf{PKW.Wrap}^{\mathsf{P}}(1^\lambda, K_u, t, h, m)$
If $b = 0$: $c \leftarrow\$ \{0,1\}^{\mathsf{PKW.cl}(\lambda,|m|)}$
$V_u \leftarrow V_u \cup \{t\}$
$W_u \leftarrow W_u \cup \{t\}$
$\boxed{W_u \leftarrow W_u \cup \{(t,h,m)\}}$
Return y

$\textsc{Wrap}(u, t, h, m)$
Require $t \notin W_u$
$\boxed{\text{Require } (t,h,m) \notin W_u}$
Require $t \notin P_u$ and $\mathsf{corr}_u = \mathsf{false}$
$c \leftarrow \mathsf{PKW.Wrap}^{\mathsf{P}}(1^\lambda, K_u, t, h, m)$
$W_u \leftarrow W_u \cup \{t\}$
$\boxed{W_u \leftarrow W_u \cup \{(t,h,m)\}}$
Return c

$\textsc{Punc}(u, t)$
Require $t \notin P_u$ and $\mathsf{corr}_u = \mathsf{false}$
$K_u \leftarrow \mathsf{PKW.Punc}^{\mathsf{P}}(1^\lambda, K_u, t)$
$P_u \leftarrow P_u \cup \{t\}$
Return // No value is returned

$\textsc{Corrupt}(u)$
Require $V_u \subseteq P_u$
Require $\mathsf{corr}_u = \mathsf{false}$
$\mathsf{corr}_u \leftarrow \mathsf{true}$
Return K_u

$\textsc{PPrim}(\mathsf{Op}, x, y)$
$y \leftarrow\$ \mathsf{P.Op}(1^\lambda, x, y : \sigma_{\mathsf{P}})$
Return y

Fig. 17. Games defining FIND$*-RCPA and FIND$*-RCPA+ security of PKW

Security. We capture security of PKW via the games in Fig. 17. FIND$*-RCPA security is captured by $\mathsf{G}^{\mathsf{find\$^*\text{-rcpa}}}$ which contains the highlighted code. FIND$*-RCPA+ security is captured by $\mathsf{G}^{\mathsf{find\$\text{-rcpa+}}}$ which contains the boxed code. Backendal, et al. [2] analyzed schemes with respect to the former, while they proposed, but did not concretely analyze schemes with respect to the latter. Both security notions capture forward security (F) and indistinguishability from random bits (IND$), while allowing real wrappings (R) in a chosen-plaintext setting (CPA). The latter adds misuse-resistance (+), by allowing tags to be repeated.

In the game, Punc allows puncturing the key on tags of the adversary's choice and Corrupt returns the current key. In the real world ($b = 1$), both RO$-Wrap and Wrap will return real wrappings while in the random world ($b = 0$) RO$-Wrap will return random ciphertexts of the appropriate length.

Sets W, V, and P and boolean corr store per-user information to prevent trivial wins. In $\mathsf{G}^{\mathsf{find\$^*\text{-rcpa}}}$ the sets store, respectively, the tags that have been used for any wrapping, the tags that have been used for challenge real-or-random wrappings, and the tags that have been punctured at. The boolean tracks whether the key has been corrupted. The game disallows repeating tags for different wrappings, making queries for previously punctured tags, corrupting keys that that have not been punctured at all tags used for challenges, and making queries for already corrupted users. Game $\mathsf{G}^{\mathsf{find\$^*\text{-rcpa}}}$ adds misuse resistance by having the set W instead stores entire (t, h, m) tuples and disallowing their reuse.

$\underline{\mathsf{PKW}[\mathsf{CF},\mathsf{NE}].\mathsf{Kg}(1^\lambda)}$
$k \leftarrow\!\!\$\ \mathsf{CF.K}(\lambda)$
$K \leftarrow \mathsf{CF.Con}^{\mathsf{P}}(1^\lambda, k, \mathsf{CF.Inp}(\lambda))$
Return $(K, \mathsf{CF.Inp}(\lambda))$

$\underline{\mathsf{PKW}[\mathsf{CF},\mathsf{NE}].\mathsf{Wrap}^{\mathsf{P}}(1^\lambda, (K_R, R), t, h, m)}$
If $t \notin R$: Return $\bot$
$sk \leftarrow \mathsf{CF.CEv}^{\mathsf{P}}(1^\lambda, K_R, R, t)$
$c \leftarrow \mathsf{NE.Enc}^{\mathsf{P}}(1^\lambda, sk, n_0, h, m)$
Return c

$\underline{\mathsf{PKW}[\mathsf{CF},\mathsf{NE}].\mathsf{Punc}^{\mathsf{P}}(1^\lambda, (K_R, R), t)}$
If $t \notin R$: Return (K_R, R)
$K_{R\setminus\{t\}} \leftarrow \mathsf{CF.CCon}^{\mathsf{P}}(1^\lambda, K_R, R, R \setminus \{t\})$
Return $(K_{R\setminus\{t\}}, R \setminus \{t\})$

$\underline{\mathsf{PKW}[\mathsf{CF},\mathsf{NE}].\mathsf{Unwrap}^{\mathsf{P}}(1^\lambda, (K_R, R), t, h, c)}$
If $t \notin R$: Return $\bot$
$sk \leftarrow \mathsf{CF.CEv}^{\mathsf{P}}(1^\lambda, K_R, R, t)$
$m \leftarrow \mathsf{NE.Dec}^{\mathsf{P}}(1^\lambda, sk, n_0, h, c)$
Return m

Fig. 18. Algorithms of puncturable key wrapping scheme $\mathsf{PKW}[\mathsf{CF},\mathsf{NE}]$

The FIND\$*-RCPA+ advantage of $\mathcal{A}$ is $\mathsf{Adv}^{\mathsf{find\$\text{-}rcpa+}}_{\mathsf{PKW},\mathsf{P},\mathcal{A}}(\lambda) = 2\Pr[\mathsf{G}^{\mathsf{find\$\text{-}rcpa+}}_{\mathsf{PKW},\mathsf{P},\mathcal{A}}(\lambda)] - 1$. If $\mathsf{Adv}^{\mathsf{find\$\text{-}rcpa+}}_{\mathsf{PKW},\mathsf{P},\mathcal{A}}$ is negligible for all PPT $\mathcal{A}$, then we say PKW is FIND\$*-RCPA+ secure with P. FIND\$*-RCPA security and its advantage function $\mathsf{Adv}^{\mathsf{find\$^*\text{-}rcpa}}_{\mathsf{PKW},\mathsf{P},\mathcal{A}}$ are defined analogously.

We made minor modifications from prior definitions for notational convenience in our proof. Backendal, et al. allowed all queries for corrupted users except for RO\$-Wrap queries. Preventing all queries for corrupted users is equivalent because the adversary can anyway use the exposed key to perfectly emulate the other oracles. Backendal, et al. allowed queries for previously punctured points unhindered (returning $\bot$ in RO\$-Wrap when $b = 0$ if it would have returned $\bot$ when $b = 1$). Preventing all queries for previously punctured points is equivalent for schemes that always return $\bot$ when wrapping/unwrapping at a previously punctured point and return the given key when puncturing a previously punctured point. We explicitly add an ideal primitive and allow adversary programming thereof to match our other definitions.

Construction. We will be analyzing the construction of Backendal, et al. [2], except we formalize it in terms of a reconstrainable CPRF rather than a puncturable PRF. Let CF be a reconstrainable constrained function family with $\mathsf{CF.SS}$ containing $\mathsf{CF.Inp} \setminus S$ for all $S \subset \mathsf{CF.Inp}$.[6] Let NE be a nonce-based encryption scheme with $\mathsf{NE.K} = \mathsf{CF.Out}$. Let n_0 denote a fixed nonce in $\mathsf{NE.N}$. Then $\mathsf{PKW}[\mathsf{CF},\mathsf{NE}]$ is defined as shown in Fig. 18. Its header space, message space, and ciphertext length function match that of NE. Its tag space is $\mathsf{CF.Inp}$.

Its key consists of a constrained key K for CF and a set R representing where the key has been constrained so far. Key generation constrains the initial CF key k on set $\mathsf{CF.Inp}(\lambda)$ and returns $(K, \mathsf{CF.Inp}(\lambda))$. We expect that $K = k$. For wrapping or unwrapping we use the output of CF on t as a key for encrypting or decrypting with NE. We puncture keys by constraining them on sets with the given tag removed.

Constrain Invariance. When analyzing PKW from the security of CF, there is a minor technical gap because the formers security definition repeatedly constrains the same key, using it for evaluations of CF and returning it for corruptions. The security definitions for CF always use the original key for evaluations and only allow "one-shot" constraining. We bridge this gap by assuming a "constrain invariance" property which captures that these views of CF are indistinguishable. Constrain invariance is captured by the game shown in Fig. 19. When $b = 1$, Con will always reconstrain keys, and CEv will evaluate using the current constrained key. When $b = 0$, a one-time constraining only happens in Corrupt, and CEv always evaluates using the root key.

The advantage of $\mathcal{A}$ is $\mathsf{Adv}^{\mathsf{coninv}^*}_{\mathsf{CF},\mathsf{P},\mathcal{A}}(\lambda) = 2\Pr[\mathsf{G}^{\mathsf{coninv}^*}_{\mathsf{CF},\mathsf{P},\mathcal{A}}(\lambda)] - 1$. If $\mathsf{Adv}^{\mathsf{coninv}^*}_{\mathsf{CF},\mathsf{P},\mathcal{A}}$ is negligible for all PPT $\mathcal{A}$, then we say CF is constrain invariant with P.

Security Result. The following theorem captures that security of PKW follows from the security of its underlying components.

[6] It is natural to build CF from a prefix-constrained function family by creating $\mathsf{CF.Inp} \setminus S$ as a union of prefix sets.

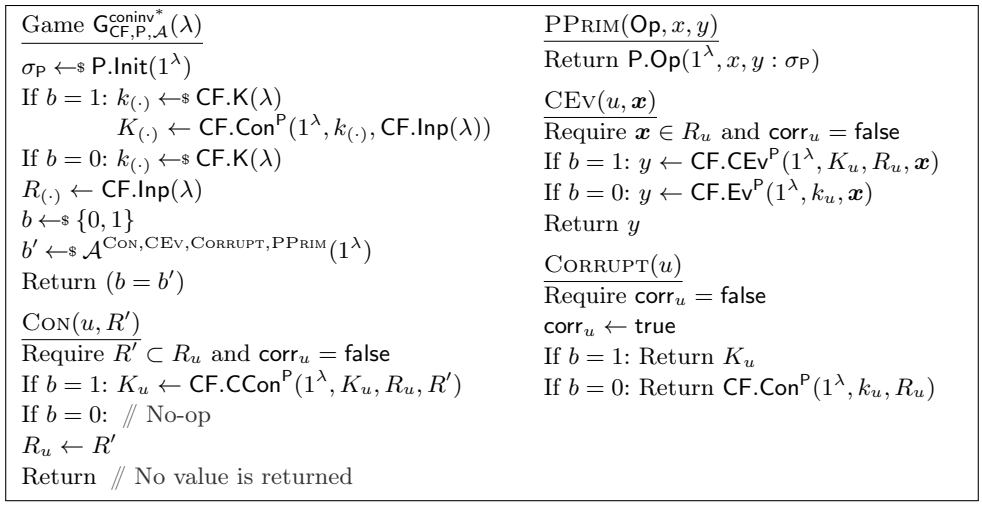


Fig. 19. Game defining constrain invariance of CF

Theorem 7. *Suppose* CF *is SIM*-AC-CPRF secure and constraint invariant with* P *and* NE *is SIM*-AC-MR\$ secure with* P*. Then* PKW[CF, NE] *is FIND\$*-RCPA+ secure with* P*.*

Proof. The proof proceeds via a series of hybrids G_0 to G_3, switching from the original security game to one where the view of the adversary is independent of the bit it is guessing. In these hybrids we first switch from repeatedly constraining the CF key to using the original key for evaluations and doing a "one-shot" constraining when a user is corrupted using constrain invariance, then switch CF from real to simulated when used in *both* wrapping oracles, then switch NE from real to simulated when used in *both* wrapping oracles. From given adversary $\mathcal{A}$, we build adversaries $\mathcal{A}_{\mathsf{coninv}^*}$, $\mathcal{A}_{\mathsf{CF}}$, and $\mathcal{A}_{\mathsf{NE}}$ for which we establish the following claims:

1. $\Pr[\mathsf{G}_0(\lambda)] = \Pr[\mathsf{G}^{\mathsf{find\$\text{-}rcpa+}}_{\mathsf{PKW[CF,NE]},\mathsf{P},\mathcal{A}}(\lambda)]$
2. $\Pr[\mathsf{G}_0(\lambda)] - \Pr[\mathsf{G}_1(\lambda)] = \mathsf{Adv}^{\mathsf{coninv}^*}_{\mathsf{CF},\mathsf{P},\mathcal{A}_{\mathsf{coninv}^*}}(\lambda)$
3. $\Pr[\mathsf{G}_1(\lambda)] - \Pr[\mathsf{G}_2(\lambda)] = \mathsf{Adv}^{\mathsf{sim}^*\text{-}\mathsf{ac}\text{-}\mathsf{cprf}}_{\mathsf{CF},\mathsf{P},\mathsf{S}_{\mathsf{CF}},\mathcal{A}_{\mathsf{CF}}}(\lambda)$
4. $\Pr[\mathsf{G}_2(\lambda)] - \Pr[\mathsf{G}_3(\lambda)] = \mathsf{Adv}^{\mathsf{sim}^*\text{-}\mathsf{ac}\text{-}\mathsf{mr\$}}_{\mathsf{NE},\mathsf{P},\mathsf{S}_{\mathsf{NE}},\mathcal{A}_{\mathsf{NE}}}(\lambda)$
5. $\Pr[\mathsf{G}_3(\lambda)] = 1/2$.

Therefore,

$$\begin{aligned}
\mathsf{Adv}^{\mathsf{find\$\text{-}rcpa+}}_{\mathsf{PKW[CF,NE]},\mathsf{P},\mathcal{A}}(\lambda) &= 2\Pr[\mathsf{G}^{\mathsf{find\$\text{-}rcpa+}}_{\mathsf{PKW[CF,NE]},\mathsf{P},\mathcal{A}}(\lambda)] - 1 \\
&= 2 \cdot \left(\sum_{i=0}^{2} (\Pr[\mathsf{G}_i(\lambda)] - \Pr[\mathsf{G}_{i+1}(\lambda)]) + \Pr[\mathsf{G}_3(\lambda)] \right) - 1 \\
&= 2 \cdot \left(\mathsf{Adv}^{\mathsf{coninv}^*}_{\mathsf{CF},\mathsf{P},\mathcal{A}_{\mathsf{coninv}^*}}(\lambda) + \mathsf{Adv}^{\mathsf{sim}^*\text{-}\mathsf{ac}\text{-}\mathsf{cprf}}_{\mathsf{CF},\mathsf{P},\mathsf{S}_{\mathsf{CF}},\mathcal{A}_{\mathsf{CF}}}(\lambda) + \mathsf{Adv}^{\mathsf{sim}^*\text{-}\mathsf{ac}\text{-}\mathsf{mr\$}}_{\mathsf{NE},\mathsf{P},\mathsf{S}_{\mathsf{NE}},\mathcal{A}_{\mathsf{NE}}}(\lambda) + 1/2 \right) - 1 \\
&= 2 \cdot \left(\mathsf{Adv}^{\mathsf{coninv}^*}_{\mathsf{CF},\mathsf{P},\mathcal{A}_{\mathsf{coninv}^*}}(\lambda) + \mathsf{Adv}^{\mathsf{sim}^*\text{-}\mathsf{ac}\text{-}\mathsf{cprf}}_{\mathsf{CF},\mathsf{P},\mathsf{S}_{\mathsf{CF}},\mathcal{A}_{\mathsf{CF}}}(\lambda) + \mathsf{Adv}^{\mathsf{sim}^*\text{-}\mathsf{ac}\text{-}\mathsf{mr\$}}_{\mathsf{NE},\mathsf{P},\mathsf{S}_{\mathsf{NE}},\mathcal{A}_{\mathsf{NE}}}(\lambda) \right).
\end{aligned}$$

The adversaries we construct are PPT (assuming given algorithms are), so this implies the result by letting S_{CF} and S_{NE} be the simulators assumed to exist by the security of CF and NE.

Claims 1 and 2. All of the hybrids are defined in Fig. 20 where comments indicate which lines of code are included in which games. We start with G_0 which was obtained by plugging the code of PKW[CF, NE]'s algorithms into $\mathsf{G}^{\mathsf{find\$\text{-}rcpa+}}_{\mathsf{PKW},\mathsf{P},\mathcal{A}}(\lambda)$. User u's key is (K_u, R_u) for which it will always hold that $R_u \sqcup P_u = \mathsf{CF.Inp}(\lambda)$,

Games $\mathsf{G}_i(\lambda)$, $i \in \{0,\ldots,3\}$

$\sigma_{\mathsf{P}} \leftarrow\$ \mathsf{P.Init}(1^\lambda)$
$k_{(\cdot)} \leftarrow\$ \mathsf{CF.K}(\lambda)$ // $i < 2$
$\sigma_{\mathsf{CF}} \leftarrow\$ \mathsf{S_{CF}.Init}(1^\lambda)$ // $2 \le i$
$sk_{(\cdot,\cdot)} \leftarrow\$ \mathsf{NE.K}(\lambda)$ // $2 \le i < 3$
$\sigma_{\mathsf{NE}} \leftarrow\$ \mathsf{S_{NE}.Init}(1^\lambda)$ // $3 \le i$
$K_{(\cdot)} \leftarrow \mathsf{CF.Con}^{\mathsf{P}}(1^\lambda, k_{(\cdot)}, \mathsf{CF.Inp}(\lambda))$ // $i < 1$
$R_{(\cdot)} \leftarrow \mathsf{CF.Inp}(\lambda)$
$b \leftarrow\$ \{0,1\}$
$b' \leftarrow\$ \mathcal{A}^{\text{RO\$-Wrap,Wrap,Punc,Corrupt,PPrim}}(1^\lambda)$
Return $(b = b')$

PPrim(Op, x, y)
$y \leftarrow\$ \mathsf{P.Op}(1^\lambda, x, y : \sigma_{\mathsf{P}})$
Return y

Wrap(u,t,h,m)
Require $(t,h,m) \notin W_u$
Require $t \in R_u$ and $\mathsf{corr}_u = \mathsf{false}$
$sk_{u,t} \leftarrow \mathsf{CF.CEv}^{\mathsf{P}}(1^\lambda, K_u, R_u, t)$ // $i < 1$
$sk_{u,t} \leftarrow \mathsf{CF.Ev}^{\mathsf{P}}(1^\lambda, k_u, t)$ // $1 \le i < 2$
$E_u \leftarrow E_u \cup \{t\}$
$c \leftarrow \mathsf{NE.Enc}^{\mathsf{P}}(1^\lambda, sk_{u,t}, n_0, h, m)$ // $i < 3$
$c \leftarrow T_{u,t}[n_0,h,m] \leftarrow\$ \{0,1\}^{\mathsf{NE.cl}(\lambda,|m|)}$ // $3 \le i$
$W_u \leftarrow W_u \cup \{(t,h,m)\}$
Return c

Punc(u,t)
Require $t \in R_u$ and $\mathsf{corr}_u = \mathsf{false}$
$K_u \leftarrow \mathsf{CF.CCon}^{\mathsf{P}}(1^\lambda, K_u, R_u, R_u \setminus \{t\})$ // $i < 1$
$R_u \leftarrow R_u \setminus \{t\}$
Return

Corrupt(u)
Require $V_u \cap R_u = \emptyset$
Require $\mathsf{corr}_u = \mathsf{false}$
$\mathsf{corr}_u \leftarrow \mathsf{true}$
$K_u \leftarrow \mathsf{CF.Con}^{\mathsf{P}}(1^\lambda, k_u, R_u)$ // $1 \le i < 2$
For $t \in R_u \cap E_u$: // $2 \le i$
  $f'(t) \leftarrow sk_{u,t}$ // $2 \le i < 3$
  $f'(t) \leftarrow\$ \mathsf{S_{NE}.Exp}^{\text{PPrim}}(1^\lambda, (u,t), T_{u,t} : \sigma_{\mathsf{NE}})$ // $3 \le i$
$K_u \leftarrow \mathsf{S_{CF}.Con}^{\text{PPrim}}(1^\lambda, u, R_u, f' : \sigma_{\mathsf{CF}})$ // $2 \le i$
Return (K_u, R_u)

RO\$-Wrap(u,t,h,m)
Require $(t,h,m) \notin W_u$
Require $t \in R_u$ and $\mathsf{corr}_u = \mathsf{false}$
If $b = 1$: $sk_{u,t} \leftarrow \mathsf{CF.CEv}^{\mathsf{P}}(1^\lambda, K_u, R_u, t)$ // $i < 1$
  $sk_{u,t} \leftarrow \mathsf{CF.Ev}^{\mathsf{P}}(1^\lambda, k_u, t)$ // $1 \le i < 2$
  $E_u \leftarrow E_u \cup \{t\}$
  $c \leftarrow \mathsf{NE.Enc}^{\mathsf{P}}(1^\lambda, sk_{u,t}, n_0, h, m)$ // $i < 3$
  $c \leftarrow T_{u,t}[n_0,h,m] \leftarrow\$ \{0,1\}^{\mathsf{NE.cl}(\lambda,|m|)}$ // $3 \le i$
If $b = 0$: $c \leftarrow\$ \{0,1\}^{\mathsf{NE.cl}(\lambda,|m|)}$
$V_u \leftarrow V_u \cup \{t\}$
$W_u \leftarrow W_u \cup \{(t,h,m)\}$
Return y

Fig. 20. Hybrids G_0 through G_3 for analyzing $\mathsf{PKW}[\mathsf{CF},\mathsf{NE}]$. Comments indicate which hybrids a line is included in. Note that for a transition into hybrid G_j, comments that end with j are added and comments that being with j are removed.

Adversary $\mathcal{A}^{\text{Con,Corrupt,CEv,PPrim}}_{\text{coninv}^*}(1^\lambda)$
$R_{(\cdot)} \leftarrow \mathsf{CF.Inp}(\lambda)$
$b \leftarrow\$ \{0,1\}$
$b' \leftarrow\$ \mathcal{A}^{\text{RO\$-Wrap,Wrap,Punc,Corrupt}',\text{PPrim}}(1^\lambda)$
If $(b = b')$: Return 1
Return 0

RO\$-Wrap(u,t,h,m)
Require $(t,h,m) \notin W_u$
Require $t \in R_u$ and $\mathsf{corr}_u = \mathsf{false}$
If $b = 1$: $sk \leftarrow$ CEv(u,t)
  $c \leftarrow \mathsf{NE.Enc}^{\mathsf{P}}(1^\lambda, sk, n_0, h, m)$
If $b = 0$: $c \leftarrow\$ \{0,1\}^{\mathsf{NE.cl}(\lambda,|m|)}$
$V_u \leftarrow V_u \cup \{t\}$
$W_u \leftarrow W_u \cup \{(t,h,m)\}$
Return y

Wrap(u,t,h,m)
Require $(t,h,m) \notin W_u$
Require $t \in R_u$ and $\mathsf{corr}_u = \mathsf{false}$
$sk \leftarrow$ CEv(u,t)
$c \leftarrow \mathsf{NE.Enc}^{\mathsf{P}}(1^\lambda, sk, n_0, h, m)$
$W_u \leftarrow W_u \cup \{(t,h,m)\}$
Return c

Punc(u,t)
Require $t \in R_u$ and $\mathsf{corr}_u = \mathsf{false}$
Con$(u, R_u \setminus \{t\})$
$R_u \leftarrow R_u \setminus \{t\}$
Return

Corrupt$'(u)$
Require $V_u \cap R_u = \emptyset$
Require $\mathsf{corr}_u = \mathsf{false}$
$\mathsf{corr}_u \leftarrow \mathsf{true}$
$K_u \leftarrow$ Corrupt(u)
Return (K_u, R_u)

Fig. 21. Constrain invariance adversary $\mathcal{A}_{\text{coninv}^*}$ used to switch from G_0 to G_1 in analysis of $\mathsf{PKW}[\mathsf{CF},\mathsf{NE}]$

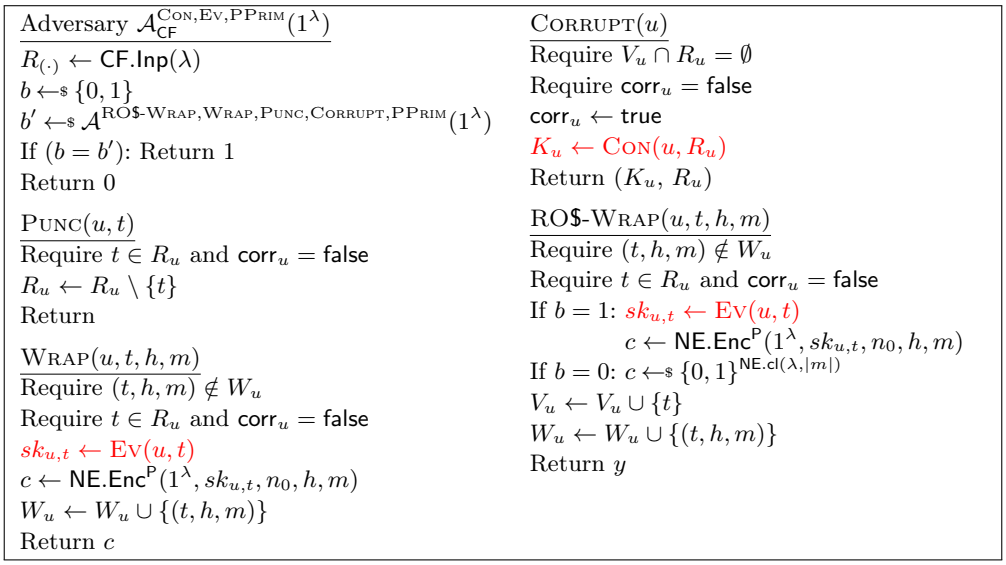


Fig. 22. SIM*-AC-CPRF adversary $\mathcal{A}_{\mathsf{CF}}$ used to switch from G_1 to G_2

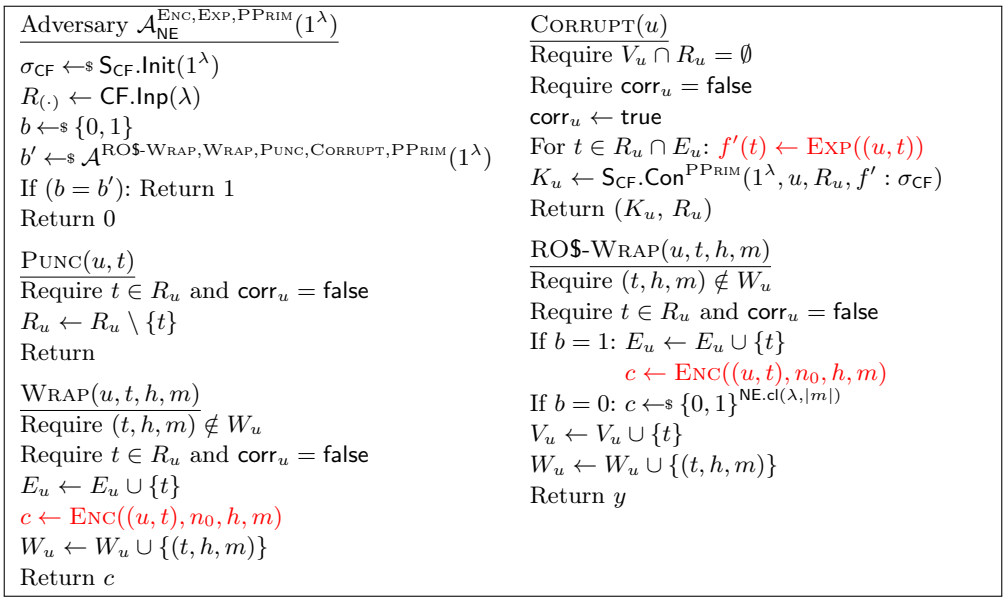


Fig. 23. SIM*-AC-MR\$ adversary $\mathcal{A}_{\mathsf{NE}}$ used to switch from G_2 to G_3 in analysis of $\mathsf{PKW}[\mathsf{CF},\mathsf{NE}]$

so we replace all uses of P with R. We added set E_u to track the inputs that CF is evaluated on. Hence our first claim, that $\Pr[\mathsf{G}_0(\lambda)] = \Pr[\mathsf{G}^{\mathsf{find\$\text{-}rcpa+}}_{\mathsf{PKW}[\mathsf{CF},\mathsf{NE}],\mathsf{P},\mathcal{A}}(\lambda)]$, follows.

Hybrid G_1 switches the usage of $\mathsf{CF.CCon}$ and $\mathsf{CF.CEv}$ to $\mathsf{CF.Con}$ and $\mathsf{CF.Ev}$. Naturally, we bound the difference between G_0 and G_1 by constrain invariance of CF. The relevant adversary $\mathcal{A}_{\mathsf{coninv}^*}$ is given in Fig. 21. It simulates these games by querying its oracles for all uses of CF. The requirements of $\mathcal{A}$'s oracles ensure that $\mathcal{A}_{\mathsf{coninv}^*}$ satisfies the requirements of its own oracles. Correctness of this simulation gives our second claim, that $\Pr[\mathsf{G}_0(\lambda)] - \Pr[\mathsf{G}_1(\lambda)] = \mathsf{Adv}^{\mathsf{coninv}^*}_{\mathsf{CF},\mathsf{P},\mathcal{A}_{\mathsf{coninv}^*}}(\lambda)$.

Claim 3. Hybrid G_2 switches the output of CF to random (and its keys to simulated). Naturally, we bound the difference between G_1 and G_2 by SIM*-AC-CPRF security of CF. The relevant adversary $\mathcal{A}_{\mathsf{CF}}$ is given in Fig. 22. It simulates these games by querying its oracles for all uses of CF. The requirements of $\mathcal{A}$'s oracles ensure that $\mathcal{A}_{\mathsf{CF}}$ only makes queries for u with $\mathsf{C}_u = \emptyset$, which allowed us to simplify code in G_2. The game samples the sk's as the beginning of the game and uses them to define the restricted function f' in CORRUPT when needed. Correctness of this simulation gives our third claim, that $\Pr[\mathsf{G}_1(\lambda)] - \Pr[\mathsf{G}_2(\lambda)] = \mathsf{Adv}^{\mathsf{sim}^*\text{-}\mathsf{ac}\text{-}\mathsf{cprf}}_{\mathsf{CF},\mathsf{P},\mathsf{S}_{\mathsf{CF}},\mathcal{A}_{\mathsf{CF}}}(\lambda)$.

Claim 4. Hybrid G_3 switches the output of NE to random (and its keys to simulated). Naturally, we bound the difference between G_2 and G_3 by SIM*-AC-MR$ security of CF. The relevant adversary $\mathcal{A}_{\mathsf{NE}}$ is given in Fig. 23. It simulates these games by querying its oracles for all uses of NE, where key $sk_{u,t}$ corresponds to a user (u,t). The requirements of $\mathcal{A}$'s oracles ensure that $\mathcal{A}_{\mathsf{NE}}$ only makes queries for which the ciphertext will be chosen at random (the user is not corrupted and the query repeated), which allowed us to simplify code regarding T in G_3. Correctness of this simulation gives our fourth claim, that $\Pr[\mathsf{G}_2(\lambda)] - \Pr[\mathsf{G}_3(\lambda)] = \mathsf{Adv}^{\mathsf{sim}^*\text{-}\mathsf{ac}\text{-}\mathsf{mr\$}}_{\mathsf{NE},\mathsf{P},\mathsf{S}_{\mathsf{NE}},\mathcal{A}_{\mathsf{NE}}}(\lambda)$.

Claim 5. Hybrid G_3 only uses b in RO$-WRAP. The ciphertext c is uniformly random and independent of b, but when $b = 1$ the tag t gets added to E_u and table $T_{u,t}$ is modified. These variables could only be used in the for loop of CORRUPT. So because that oracle ensures $V_u \cap R_u = \emptyset$, we will have $t \notin R_u$ for such tags. Hence the differing behavior based on b cannot affect the view of $\mathcal{A}$. This gives our fifth claim, that $\Pr[\mathsf{G}_3(\lambda)] = 1/2$. □

D Proofs for Hashed Constrained PRF(s)

D.1 Proof of Theorem 2 (Hashed[CF] is SIM*-AC-CPRF secure)

Proof. Our proof will proceed in a series of hybrids. We construct a series of hybrids G_0 to G_3 switching from the real world of SIM*-AC-CPRF to the simulated world. The first transition (G_0 to G_1) is bounded by a bad event that $\mathsf{CF.Ev}(k,x) = y$ inside of a $\textsc{Ev}(u,x)$ query and while the x has not been constrained for u, the adversary manages to query the random oracle on y or a different EV query that also computes y from $\mathsf{CF.Ev}(k',x')$. The difference between these games is bounded by the advantage of an IND*-NC-CPRF adversary against CF because these events will be unlikely if y is random. IND*-NC-CPRF security suffices because the adversary will guess (u,x) and query for it at the start of the game. This required that P be non-programmable so that making this query at the start of the game did not change the distribution of EV. Game G_2 is a syntactic rewriting of G_1 to prepare for the transition to G_3. This latter transition is bounded by a bad event that two different executions of $\mathsf{CF.Ev}(k,x)$ for an unexposed user gives different results. If we assumed that P was consistent (an eminently reasonable assumption), then the probability of this would be zero. Without that assumption, we instead make a reduction to the IND*-PRF security of CF that repeatedly queries for all (u,x) pairs and checks whether the outputs are consistent. (Here we are using that the random world of IND*-PRF security always returns consistent random values.) Now in G_3 we have reached a game that can be captured by the simulated world of SIM*-AC-CPRF with a simulator that attempts to reprogram the random oracle appropriately on constrain queries and otherwise simply runs the honest algorithms of CF.

We will call our simulator S. Let $\mathcal{A}_{\mathsf{cprf}^*}$ be the given adversary against $\mathsf{Hashed}[\mathsf{CF}]$. Assume it makes at most $p(\lambda)$ queries to PPRIM and $q(\lambda)$ queries to EV. Assume that $N(\lambda)$ is a set such that whenever $\mathcal{A}_{\mathsf{cprf}^*}$

queries $\text{Ev}(u,x)$ it holds that $(u,x) \in N$. From $\mathcal{A}_{\mathsf{cprf}^*}$ (and N) we will construct $\mathcal{A}_{\mathsf{cprf}}$ and $\mathcal{A}_{\mathsf{prf}}$. Then the following claims will hold:

1. $\Pr[\mathsf{G}_0(\lambda)] = \Pr[\mathsf{G}^{\mathsf{sim}^*\text{-ac-cprf}}_{\mathsf{Hashed}[\mathsf{CF}],\mathsf{S},\mathsf{P}\times\mathsf{P}_{\mathsf{rom}},\mathcal{A}_{\mathsf{cprf}^*}}(\lambda)|b=1]$
2. $\Pr[\mathsf{G}_0(\lambda)] - \Pr[\mathsf{G}_1(\lambda)] \leq |N(\lambda)| \cdot (\mathsf{Adv}^{\mathsf{ind}^*\text{-nc-cprf}}_{\mathsf{CF},\mathsf{P},\mathcal{A}_{\mathsf{cprf}}}(\lambda) + \frac{q(\lambda)+p(\lambda)}{\min_x |\mathsf{CF.Out}(\lambda,x)|})$
3. $\Pr[\mathsf{G}_1(\lambda)] = \Pr[\mathsf{G}_2(\lambda)]$
4. $\Pr[\mathsf{G}_2(\lambda)] - \Pr[\mathsf{G}_3(\lambda)] \leq \mathsf{Adv}^{\mathsf{ind}^*\text{-prf}}_{\mathsf{CF},\mathsf{P},\mathcal{A}_{\mathsf{prf}}}(\lambda)$
5. $\Pr[\mathsf{G}_3(\lambda)] = \Pr[\neg\mathsf{G}^{\mathsf{sim}^*\text{-ac-cprf}}_{\mathsf{Hashed}[\mathsf{CF}],\mathsf{S},\mathsf{P}\times\mathsf{P}_{\mathsf{rom}},\mathcal{A}_{\mathsf{cprf}^*}}(\lambda)|b=0]$.

Putting these claims together gives

$$\begin{aligned}
&\mathsf{Adv}^{\mathsf{sim}^*\text{-ac-cprf}}_{\mathsf{Hashed}[\mathsf{CF}],u,\mathsf{S},\mathsf{P}\times\mathsf{P}_{\mathsf{rom}},\mathcal{A}_{\mathsf{cprf}^*}}(\lambda) = \Pr[\mathsf{G}_0(\lambda)] - \Pr[\mathsf{G}_3(\lambda)] \\
&= \Pr[\mathsf{G}_0(\lambda)] - \Pr[\mathsf{G}_1(\lambda)] + \Pr[\mathsf{G}_1(\lambda)] - \Pr[\mathsf{G}_2(\lambda)] + \Pr[\mathsf{G}_2(\lambda)] - \Pr[\mathsf{G}_3(\lambda)] \\
&\leq |N(\lambda)| \cdot \mathsf{Adv}^{\mathsf{ind}^*\text{-nc-cprf}}_{\mathsf{CF},\mathsf{P},\mathcal{A}_{\mathsf{cprf}}}(\lambda) + \frac{|N(\lambda)|(q(\lambda)+p(\lambda))}{\min_x |\mathsf{CF.Out}(\lambda,x)|} + \mathsf{Adv}^{\mathsf{ind}^*\text{-prf}}_{\mathsf{CF},\mathsf{P},\mathcal{A}_{\mathsf{prf}}}(\lambda).
\end{aligned}$$

Then result holds from $|N(\lambda)|$, $q(\lambda)$, and $p(\lambda)$ being polynomial; $\min_x |\mathsf{CF.Out}(\lambda,x)|$ being super-polynomial; and the assumed security of CF. (Note that IND*-NC-CPRF security directly implies IND*-PRF security.) If P is consistent, then $\mathsf{Adv}^{\mathsf{ind}^*\text{-prf}}_{\mathsf{CF},\mathsf{P},\mathcal{A}_{\mathsf{prf}}}(\lambda) = 0$.

Claim 1. Game G_0 is defined in Fig. 24. It contains highlighted code. Throughout we use x to refer to inputs to CF, y to refer to outputs of CF or inputs to the random oracle, and z to refer to outputs of the random oracle (which are often implicitly outputs of $\mathsf{Hashed}[\mathsf{CF}]$).

This game makes explicit the random sampling of the random oracle using table σ_{rom} and moreover splits the storing of values in the random oracle to also use tables $T_{u,x}$ (in preparation for the coming game transitions). We store $T_{u,x}[y] \leftarrow z$ at the end of $\text{Ev}(u,x)$ when y is the output computed for CF and z the output of the random oracle. For unconstrained inputs ($x \notin C_u$), we attempt to use $T_{u,x}$ for our lazy sampling of the random oracle. To prevent inconsistencies with the random oracle as presented by PPRIM we check if $\sigma_{\mathsf{rom}}[y]$ has a value and use that instead, if so. To prevent inconsistencies with Ev queries for any other inputs — $(v,w) \neq (u,x)$ — which computed the same y we check if $T_{v,w}[y]$ has a value and use that instead, if so. For constrained inputs ($x \in C_u$) we call PPRIM to use σ_{rom}.

Inside PPRIM we similarly attempt to lazily sample the random oracle using σ_{rom}. To prevent inconsistencies with Ev queries for unconstrained inputs we check if any $T_{u,x}[y]$ has a value for $x \notin C_u$ and use that instead, if so. When an earlier Ev input (u,x) is first constrained, we till program σ_{rom} to be consistent with $T_{u,x}$.

That this correctly simulates the real SIM*-AC-CPRF world (i.e., the first claim holds) follows from observing that the following properties will be invariant.

- Any combination of the $T_{u,x}$ tables and/or σ_{rom} which contain non-$\perp$ values all contain the same value.
- Once a table entry for y is set, it will never be changed.
- The first time a table entry for y is set, it is set appropriately (randomly in Ev or a lazy sampling query to PPRIM, non-randomly in a programming query to PPRIM).
- If $x \in C_u$ and $T_{u,x}[y] \neq \perp$, then $\sigma_{\mathsf{rom}}[y] \neq \perp$.

Claim 2. Game G_1 is defined in Fig. 24. It does not contain highlighted code. The removed code contains all of the table rewriting to preserve consistency of the random oracle across different types of queries. It is syntactically clear that G_0 and G_1 are identical until a $\mathsf{bad}_{u,x}$ flag is set. The fundamental lemma of game playing [8] gives $\Pr[\mathsf{G}_0(\lambda)] - \Pr[\mathsf{G}_1(\lambda)] \leq \Pr[\bigvee_{(U,X)\in N(\lambda)} \mathsf{G}_1(\lambda) \text{ sets } \mathsf{bad}_{U,X}]$.

Consider $\mathsf{bad}_{U,X}$ for fixed U, X. Setting this flag in PPRIM requires the adversary to query the random oracle on a $Y = \mathsf{CF.Ev}(k_U, X)$ value while X has not been constrained for U. The first place the flag could be set in Ev is the same, except that the random oracle query happened first. The second place the flag

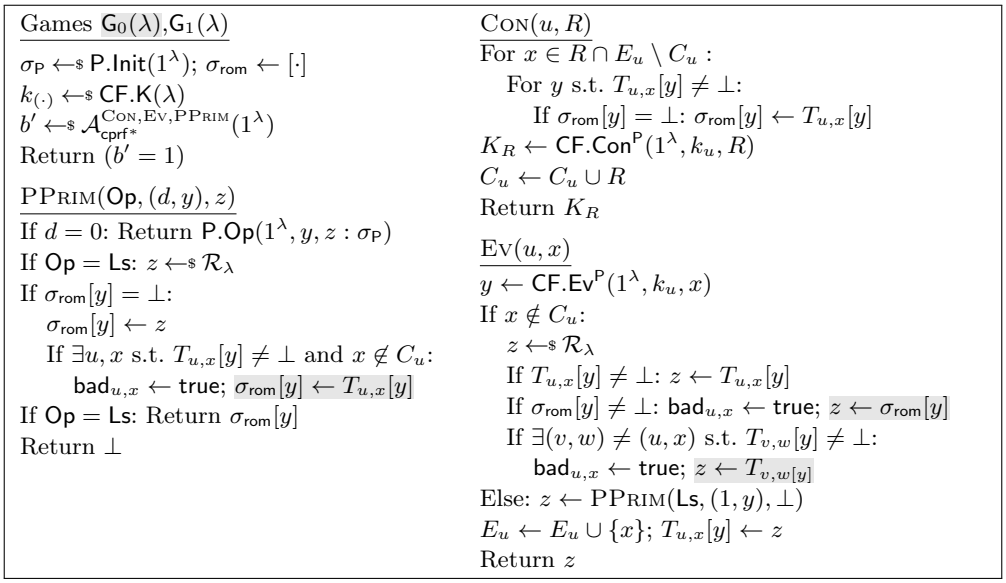


Fig. 24. Hybrids G_0 and G_1 for analyzing $\mathsf{Hashed}[\mathsf{CF}]$

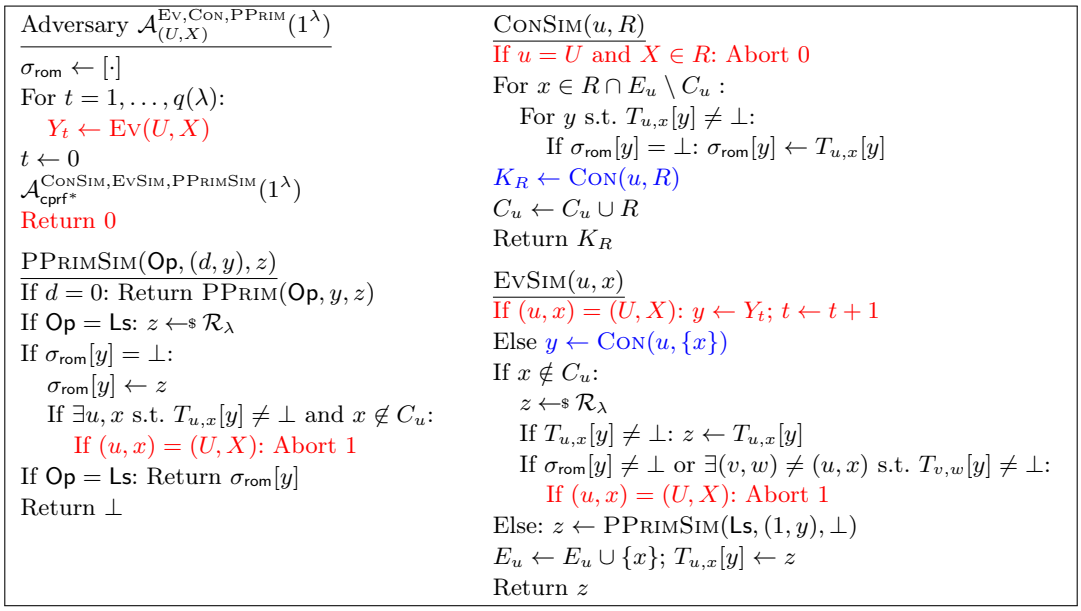


Fig. 25. Non-adaptive IND*-NC-CPRF adversary $\mathcal{A}_{(U,X)}$ for bounding the probability $\mathsf{bad}_{U,X}$ is set in G_1. Abort statements indicate that $\mathcal{A}_{U,X}$ will halt immediately with the specified output.

could be set in Ev requires a Ev query for a different input to have calculated the same Y. In all these cases, Y should look uniformly random so the probability of these events should be small.

We formalize this idea with the IND*-NC-CPRF adversary $\mathcal{A}_{(U,X)}$ shown in Fig. 25. It attempts to simulate G_1 with the goal of outputting 1 whenever it would set $\mathsf{bad}_{U,X}$ and outputting 0 otherwise. It simulates the random oracle for itself and uses its oracle to for CF. For constrain queries and evaluation queries other than for (U, X) it uses its Con oracle. (For the latter we are using our convention that $\mathsf{CF.Con}(k, \{x\}) = \mathsf{CF.Ev}(k, x)$.) For simulating evaluation queries on (U, X) it precomputes a list of $\mathsf{CF.Ev}$ outputs at the beginning of execution using Ev and then uses the values from this list to respond to later queries. Note that we can only precompute these values because P is non-programmable. If we assumed P was consistent, then $\mathsf{CF.Ev}$ would be too and we would only have needed to make a single Ev query. If $\mathsf{bad}_{U,X}$ would ever be set, $\mathcal{A}_{(U,X)}$ aborts early and returns 1. It cannot simulate constrain queries that include (U, X) so, as $\mathsf{bad}_{U,X}$ can no longer be set after such a query, it aborts early and returns 0.

When $\mathcal{A}_{(U,X)}$'s oracles are honestly running CF, it correctly simulates G_1 and so $\Pr[\mathsf{G}_1(\lambda) \text{ sets } \mathsf{bad}_{U,X}] = \Pr[\mathsf{G}^{\mathsf{ind^*\text{-}ac\text{-}cprf}}_{\mathsf{CF},\mathsf{P},\mathcal{A}_{(U,X)}}(\lambda) | b = 1]$. Consider when its oracles are simulated. The Y_t's are uniformly random and all equal to each other. Each query to the random oracle query (whether directly or through an evaluation query with $x \in C_u$) and each evaluation query for a different input with $x \notin C_u$ can be viewed as giving a single guess to the value of Y_1. There are at most $q(\lambda)+p(\lambda)$ such queries, each of which have an at most $1/|\mathsf{CF.Out}(\lambda, X)|$ probability of being correct so a union bound gives $\Pr[\neg \mathsf{G}^{\mathsf{ind^*\text{-}ac\text{-}cprf}}_{\mathsf{CF},\mathsf{P},\mathcal{A}_{(U,X)}}(\lambda) | b = 0] \leq (q(\lambda)+p(\lambda))/|\mathsf{CF.Out}(\lambda, X)|$. Hence $\Pr[\mathsf{G}_1(\lambda) \text{ sets } \mathsf{bad}_{U,X}] \leq \mathsf{Adv}^{\mathsf{ind^*\text{-}nc\text{-}cprf}}_{\mathsf{CF},\mathsf{P},\mathcal{A}_{U,X}}(\lambda) + (q(\lambda) + p(\lambda))/|\mathsf{CF.Out}(\lambda, X)|$.

A union bound over all of the possible (U, X) values in $N(\lambda)$ gives that $\Pr[\mathsf{G}_0(\lambda)] - \Pr[\mathsf{G}_1(\lambda)] \leq \left(\sum_{U,X} \mathsf{Adv}^{\mathsf{ind^*\text{-}nc\text{-}cprf}}_{\mathsf{CF},\mathsf{P},\mathcal{A}_{U,X}}(\lambda) + (q(\lambda) + p(\lambda))/|\mathsf{CF.Out}(\lambda, X)|\right)$. Now letting $\mathcal{A}_{\mathsf{cprf}}$ be the adversary that samples $(U, X) \leftarrow\!\!\$\ N(\lambda)$ then runs $\mathcal{A}_{U,X}$ gives us the second claim $\Pr[\mathsf{G}_0(\lambda)] - \Pr[\mathsf{G}_1(\lambda)] \leq |N(\lambda)| \cdot (\mathsf{Adv}^{\mathsf{ind^*\text{-}nc\text{-}cprf}}_{\mathsf{CF},\mathsf{P},\mathcal{A}_{\mathsf{cprf}}}(\lambda) + \frac{q(\lambda)+p(\lambda)}{\min_x |\mathsf{CF.Out}(\lambda,x)|})$.

Claim 3. Game G_1 is in some senses "most of the way" to the conclusion of the proof. However, in this game $\text{Ev}(u, x)$ may return different results across repeated queries for inputs that have not been constrained if different y's are computed inside of them. However, the simulated world of SIM*-AC-CPRF ensures repeated queries are answered consistently for such inputs. Our next pair of games will switch from using tables $T_{u,x}[y]$ to using $T_u[x]$, removing the dependence on y that causes inconsistent behavior.

Consider G_2 defined in Fig. 26. It contains highlighted but not boxed code. In Ev, we use $T_u[x]$ in place of $T_{u,x}[y]$ if they are equal. If they are unequal, we set bad and use $T_{u,x}$ as before. In Con, we break down the reprogramming of the random oracle. First we compute $Y = \mathsf{CF.Ev}(k_u, x)$. (Non-programmability of P ensures this does not change the adversary's view.) If the set of y's computed in prior $\text{Ev}(u, x)$ queries (computed by checking if $T_{u,x}[y]$ is empty) is not a singleton set containing only Y, we set bad and program σ_{rom} as before. Otherwise, if $T_u[x]$ and $T_{u,x}[Y]$ are unequal, we set bad and program σ_{rom} using $T_{u,x}[Y]$ as before. If both of those pass, we program σ_{rom} using $T_u[x]$.

This logic ensures that $T_u[x]$ and Y are used precisely when they give the same result as would have been obtained before (and otherwise we fall back to not using them). Consequently, G_2 is equivalent to G_1 giving the third claim that $\Pr[\mathsf{G}_1(\lambda)] = \Pr[\mathsf{G}_2(\lambda)]$.

Claim 4. Consider G_3 also in Fig. 26 which contains boxed but not highlighted code. Whenever G_2 would fall back on acting like G_1 because using $T_u[x]$ and Y would behave incorrectly, G_3 just uses them anyway. It is syntactically clear that G_2 and G_3 are identical until the bad flag is set. Hence, $\Pr[\mathsf{G}_2(\lambda)] - \Pr[\mathsf{G}_3(\lambda)] \leq \Pr[\mathsf{G}_3(\lambda) \text{ sets } \mathsf{bad}]$.

It is not hard to verify that if for all (u, x) executing $\mathsf{CF.Ev}(k_u, x)$ always returns the same value, then bad will never be set. So we can bound the probability of bad by the probability we ever see inconsistent outputs of $\mathsf{CF.Ev}$. If P is consistent, the probability of this is 0. Otherwise, we can reduce this to the IND*-PRF security of CF. This is captured by the adversary $\mathcal{A}_{\mathsf{prf}}$ given in Fig. 27. It simply makes $q(\lambda) + 1$ queries for each possible Ev query and outputs 1 if it sees any inconsistency. Because P is non-programmable, querying Ev at the beginning of the game does not change the distribution of its outputs and

Games $\mathsf{G}_2(\lambda)$, $\boxed{\mathsf{G}_3(\lambda)}$

$\sigma_{\mathsf{P}} \leftarrow\$ \mathsf{P.Init}(1^\lambda)$; $\sigma_{\mathsf{rom}} \leftarrow [\cdot]$
$k_{(\cdot)} \leftarrow\$ \mathsf{CF.K}(\lambda)$
$b' \leftarrow\$ \mathcal{A}^{\textsc{Con},\textsc{Ev},\textsc{PPRim}}_{\mathsf{cprf}^*}(1^\lambda)$
Return $(b' = 1)$

$\textsc{PPRim}(\mathsf{Op}, (d, y), z)$
If $d = 0$: Return $\mathsf{P.Op}(1^\lambda, y, z : \sigma_{\mathsf{P}})$
If $\mathsf{Op} = \mathsf{Ls}$: $z \leftarrow\$ \mathcal{R}_\lambda$
If $\sigma_{\mathsf{rom}}[y] = \bot$: $\sigma_{\mathsf{rom}}[y] \leftarrow z$
If $\mathsf{Op} = \mathsf{Ls}$: Return $\sigma_{\mathsf{rom}}[y]$
Return $\bot$

$\textsc{Con}(u, R)$
For $x \in R \cap E_u \setminus C_u$:
  $Y \leftarrow \mathsf{CF.Ev}^{\mathsf{P}}(1^\lambda, k_u, x)$
  If $\{Y\} \neq \{y \ : \ T_{u,x}[y] \neq \bot\}$:
    $\mathsf{bad} \leftarrow \mathsf{true}$
    For y s.t. $T_{u,x}[y] \neq \bot$:
      If $\sigma_{\mathsf{rom}}[y] = \bot$: $\sigma_{\mathsf{rom}}[y] \leftarrow T_{u,x}[y]$
    $\boxed{\text{If } \sigma_{\mathsf{rom}}[Y] = \bot\text{: } \sigma_{\mathsf{rom}}[Y] \leftarrow T_u[x]}$
  Else if $T_{u,x}[Y] \neq T_u[x]$
    $\mathsf{bad} \leftarrow \mathsf{true}$
    If $\sigma_{\mathsf{rom}}[Y] = \bot$: $\sigma_{\mathsf{rom}}[Y] \leftarrow T_{u,x}[Y]$
    $\boxed{\text{If } \sigma_{\mathsf{rom}}[Y] = \bot\text{: } \sigma_{\mathsf{rom}}[Y] \leftarrow T_u[x]}$
  Else if $\sigma_{\mathsf{rom}}[Y] = \bot$: $\sigma_{\mathsf{rom}}[Y] \leftarrow T_u[x]$
$K_R \leftarrow \mathsf{CF.Con}^{\mathsf{P}}(1^\lambda, k_u, R)$
$C_u \leftarrow C_u \cup R$
Return K_R

$\textsc{Ev}(u, x)$
$y \leftarrow \mathsf{CF.Ev}^{\mathsf{P}}(1^\lambda, k_u, x)$
If $x \notin C_u$:
  $z \leftarrow\$ \mathcal{R}_\lambda$
  If $T_{u,x}[y] \neq T_u[x]$
    $\mathsf{bad} \leftarrow \mathsf{true}$
    If $T_{u,x}[y] \neq \bot$: $z \leftarrow T_{u,x}[y]$
    $\boxed{\text{If } T_u[x] \neq \bot\text{: } z \leftarrow T_u[x]}$
  Else if $T_u[x] \neq \bot$: $z \leftarrow T_u[x]$
Else: $z \leftarrow \textsc{PPRim}(\mathsf{Ls}, (1, y), \bot)$
$E_u \leftarrow E_u \cup \{x\}$
$T_{u,x}[y] \leftarrow z$; $T_u[x] \leftarrow z$
Return z

Fig. 26. Hybrids G_2 and G_3 for analyzing $\mathsf{Hashed}[\mathsf{CF}]$

Adversary $\mathcal{A}^{\textsc{Ev},\textsc{PPRim}}_{\mathsf{prf}}(1^\lambda)$
For $(u, x) \in N(\lambda)$:
  For $i = 1, \ldots, q(\lambda) + 1$:
    $y_{u,x,i} \leftarrow \textsc{Ev}(u, x)$
If $\exists u, x, i, j$ s.t. $y_{u,x,i} \neq y_{u,x,j}$:
  Return 1
Return 0

Fig. 27. IND*-PRF adversary $\mathcal{A}_{\mathsf{prf}}$ for bounding the probability of bad in G_3

$\mathsf{S.Init}()$
$k_{(\cdot)} \leftarrow\$ \mathsf{CF.K}(\lambda)$
Return $k_{(\cdot)}$

$\mathsf{S.Ev}^{\textsc{PPRim}}(u, x : k)$
$y \leftarrow \mathsf{Hashed}[\mathsf{CF}].\mathsf{Ev}^{\mathsf{P}\times\mathsf{P}_{\mathsf{rom}}}(1^\lambda, k_u, x)$
Return y

$\mathsf{S.Con}^{\textsc{PPRim}}(u, R, f : k)$
For $x \in f.\mathsf{D}$:
  $Y \leftarrow \mathsf{CF.Ev}^{\mathsf{P}}(1^\lambda, k_u, x)$
  $\textsc{PPRim}(\mathsf{Prog}, (1, Y, f(x)))$
Return $\mathsf{CF.Con}^{\mathsf{P}}(1^\lambda, k, R)$

Fig. 28. Simulator S for $\mathsf{Hashed}[\mathsf{CF}]$

Games $\mathsf{G}_4(\lambda)$

$\sigma_{\mathsf{P}} \leftarrow\$ \mathsf{P.Init}(1^\lambda)$; $\sigma_{\mathsf{rom}} \leftarrow [\cdot]$
$k_{(\cdot)} \leftarrow\$ \mathsf{CF.K}(\lambda)$
$b' \leftarrow\$ \mathcal{A}^{\textsc{Con},\textsc{Ev},\textsc{PPRim}}_{\mathsf{cprf}^*}(1^\lambda)$
Return $(b' = 1)$

$\textsc{PPRim}(\mathsf{Op}, (d, y), z)$
If $d = 0$: Return $\mathsf{P.Op}(1^\lambda, y, z : \sigma_{\mathsf{P}})$
Else: Return $\mathsf{P}_{\mathsf{rom}}.\mathsf{Op}(1^\lambda, y, z : \sigma_{\mathsf{rom}})$

$\textsc{Con}(u, R)$
For $x \in R \cap E_u \setminus C_u$:
  $Y \leftarrow \mathsf{CF.Ev}^{\mathsf{P}}(1^\lambda, k_u, x)$
  $\textsc{PPRim}(\mathsf{Prog}, (1, Y), T_u[x])$
$K_R \leftarrow \mathsf{CF.Con}^{\mathsf{P}}(1^\lambda, k_u, R)$
$C_u \leftarrow C_u \cup R$
Return K_R

$\textsc{Ev}(u, x)$
$y \leftarrow \mathsf{CF.Ev}^{\mathsf{P}}(1^\lambda, k_u, x)$
If $x \notin C_u$:
  $z \leftarrow\$ \mathcal{R}_\lambda$
  If $T_u[x] \neq \bot$: $z \leftarrow T_u[x]$
Else: $z \leftarrow \textsc{PPRim}(\mathsf{Ls}, (1, y), \bot)$
$E_u \leftarrow E_u \cup \{x\}$
$T_u[x] \leftarrow z$
Return z

Fig. 29. Hybrid G_4 for analyzing $\mathsf{Hashed}[\mathsf{CF}]$

so $\Pr[\mathsf{G}_3(\lambda) \text{ sets } \mathsf{bad}] \leq \Pr[\mathsf{G}^{\mathsf{ind}^*\text{-}\mathsf{prf}}_{\mathsf{CF},\mathsf{P},\mathcal{A}_{\mathsf{prf}}}(\lambda)|b=1]$. The outputs of Ev are forced to be consistent in the random world, so $\Pr[\neg\mathsf{G}^{\mathsf{ind}^*\text{-}\mathsf{prf}}_{\mathsf{CF},\mathsf{P},\mathcal{A}_{\mathsf{prf}}}(\lambda)|b=0]=0$. This gives, $\Pr[\mathsf{G}_3(\lambda) \text{ sets } \mathsf{bad}] \leq \mathsf{Adv}^{\mathsf{ind}^*\text{-}\mathsf{prf}}_{\mathsf{CF},\mathsf{P},\mathcal{A}_{\mathsf{prf}}}(\lambda)$ and hence the claim.

Claim 5. For the final claim, first consider game G_4 in Fig. 29. It was obtained by copying G_3, simplying the if statements that are identical in all branches, simplifying PPrim, and replacing explicit programming of σ_{rom} in Con with the equivalent query to PPrim. We have $\Pr[\mathsf{G}_3(\lambda)] = \Pr[\mathsf{G}_4(\lambda)]$.

Now we can naturally see that this game is identical to the simulated SIM*-AC-CPRF world if we use the simulator given in Fig. 28. The function f_u is lazily sampled as table T_u. There are extra $\mathsf{CF.Ev}$ queries in Ev for unconstrained inputs, but non-programmability of P ensures this does not change the view. The simulator is quite natural, it runs the honest evaluation for constrained inputs. For generating constrained keys, it first tries to program the random oracle to be consistent with prior evaluations and then runs the honest constrain algorithm. The fifth claim holds, $\Pr[\mathsf{G}_3(\lambda)] = \Pr[\neg\mathsf{G}^{\mathsf{sim}^*\text{-}\mathsf{ac}\text{-}\mathsf{cprf}}_{\mathsf{Hashed}[\mathsf{CF}],\mathsf{S},\mathsf{P}\times\mathsf{P}_{\mathsf{rom}},\mathcal{A}_{\mathsf{cprf}^*}}(\lambda)|b=0]$. □

Proof variant using IND*-AC-CPRF. It seems likely that if we are willing to assume IND*-AC-CPRF security rather than IND*-NC-CPRF, then the theorem would hold without requiring $N(\lambda)$ be a priori fixed. Adversary $\mathcal{A}_{\mathsf{cprf}}$ could instead guess $i \in \{1,\ldots,q(\lambda)\}$ corresponding to the first time Ev will be queried for a (U,X) that causes the bad event, then query Ev for calculating $\mathsf{CF.Ev}(U,X)$ then and all future times it is needed. It can check whether $\mathsf{bad}_{U,X}$ has been set at the end of the game.

Similarly for $\mathcal{A}_{\mathsf{prf}}$, it could guess the first time Ev will be queried for a (U,X) that will cause bad by giving inconsistent responses. It can then immediately make the $q(\lambda)+1$ queries for (U,X) and check for collisions.

In both of these reductions, the adaptivity should let it simulate the view of $\mathcal{A}_{\mathsf{cprf}^*}$ before knowing what will be queried to Ev.

D.2 Proof of Theorem 3 (TDCF is SIM*-AC-CPRF secure)

Trapdoor Permutations. We start by providing formal syntax for a trapdoor permutation and writing the code of TDCF in terms of this syntax. We recall the standard security definitions for a trapdoor permutation.

A trapdoor permutation Π specifies key generation algorithm $\Pi.\mathsf{Kg}$, evaluation algorithm $\Pi.\mathsf{Ev}$, and inverse algorithm $\Pi.\mathsf{Inv}$. It also specifies sets $\Pi.\mathsf{Inp}$. Their syntaxes are $(tk,pub) \leftarrow\!\!\$\ \Pi.\mathsf{Kg}(1^\lambda)$, $ST' \leftarrow \Pi.\mathsf{Ev}^{\mathsf{P}}(1^\lambda,pub,ST)$, and $ST \leftarrow \Pi.\mathsf{Inv}^{\mathsf{P}}(1^\lambda,tk,ST')$. Here $ST,ST' \in \Pi.\mathsf{Inp}(\lambda,pub)$. We often write $\Pi^n_{pub}(\cdot)$ for the n-fold repetition of $\Pi.\mathsf{Ev}^{\mathsf{P}}(1^\lambda,pub,\cdot)$ and $\Pi^{-n}_{tk}(\cdot)$ for the n-fold repetition of $\Pi.\mathsf{Inv}^{\mathsf{P}}(1^\lambda,tk,\cdot)$. Correctness implies that Π_{pub} is a permutation with inverse Π^{-1}_{tk}.

One-wayness security of Π is defined as follows where PPrim is defined in the usual way:

$$\mathsf{Adv}^{\mathsf{ow}^*}_{\Pi,\mathsf{P},\mathcal{A}}(\lambda) = \Pr[ST = \mathcal{A}^{\mathrm{PP}_{\mathrm{RIM}}}(1^\lambda,pub,\Pi_{pub}(ST)) \ : \ (tk,pub) \leftarrow\!\!\$\ \Pi.\mathsf{Kg}(1^\lambda); ST \leftarrow\!\!\$\ \Pi.\mathsf{Inp}(\lambda,pub)].$$

If $\mathsf{Adv}^{\mathsf{ow}^*}_{\Pi,\mathsf{P},\mathcal{A}}$ is negligible for all PPT $\mathcal{A}$, we say that Π is OW* secure with P.

Analysis. Now we proceed to the actual proof of Theorem 3.

Proof. Recall that Π can be expressed as $\mathsf{Hashed}[\mathsf{CF}]$ where $\mathsf{CF.Kg}(1^\lambda)$ samples $(tk,pub) \leftarrow\!\!\$\ \Pi.\mathsf{Kg}(1^\lambda)$ and $ST_0 \leftarrow\!\!\$\ \Pi.\mathsf{Inp}(\lambda,pub)$, then returns $k=(tk,pub,ST_0)$ and its algorithms are defined as

$$\begin{aligned}
\mathsf{CF.Ev}(1^\lambda,k,1^c) &= \Pi^{-c}_{tk}(ST_0)\\
\mathsf{CF.Con}(1^\lambda,k,R_n) &= (pub,\Pi^{-n}_{tk}(ST_0))\\
\mathsf{CF.CEv}(1^\lambda,(pub,ST_n)),R_n,c) &= \Pi^{n-c}_{pub}(ST_n).
\end{aligned}$$

This technically does not follow our formalism for constrained function families because its output space, $\Pi.\mathsf{Inp}(\lambda,pub)$, depends on the key and because it has a key generation algorithm instead of a keyspace. It's clear these differences are not material for the proof.

Our proofs follows from the same sequence of hybrids (replacing $\mathsf{CF.K}(\lambda)$ with $\mathsf{CF.Kg}(1^\lambda)$) and claims as the proof of Theorem 2 given in Appendix D.1 except we will replace the use of CPRF security for CF. In particular, the new claims are

Adversary $\mathcal{A}^{\text{PPRIM}}_{(U,X)}(1^\lambda, pub, ST_{|X|-1})$
$\sigma_{\mathsf{rom}} \leftarrow [\cdot]$
$k_{(\cdot)} \leftarrow\!\!\$\ \mathsf{CF.Kg}(1^\lambda)$
$k_U \leftarrow \bot$
For $i = |X|-1, \ldots, 1$:
  $ST_{i-1} \leftarrow \Pi_{pub}(ST_i)$
$S \leftarrow \emptyset$
$b' \leftarrow\!\!\$\ \mathcal{A}^{\text{CON},\text{EV},\text{PPRIMSIM}}_{\mathsf{cprf}^*}(1^\lambda)$
// Location 1
For $y \in S$:
  If $\Pi_{pub}(y) = ST_{|X|-1}$: Return y
Return $\bot$

PPRIM$(\mathsf{Op}, (d, y), z)$
If $d = 0$: Return $\mathsf{P.Op}(1^\lambda, y, z : \sigma_{\mathsf{P}})$
$S \leftarrow S \cup \{y\}$
Return $\mathsf{P}_{\mathsf{rom}}.\mathsf{Op}(1^\lambda, y, z : \sigma_{\mathsf{rom}})$

CON(u, R)
If $u = U$ and $X \in R$: Goto Location 1
For $x \in R \cap E_u \setminus C_u$:
  If $u \neq U$: $Y \leftarrow \mathsf{CF.Ev}^{\mathsf{P}}(1^\lambda, k_u, x)$
  Else: $Y \leftarrow ST_{|x|}$ // $|x| \leq |X| - 1$
  PPRIM$(\mathsf{Prog}, (1, Y), T_u[x])$
If $u \neq U$: $K_R \leftarrow \mathsf{CF.Con}^{\mathsf{P}}(1^\lambda, k_u, R)$
Else $K_R \leftarrow (pub, ST_{|R|-1})$
// $|R| - 1 \leq |X| - 1$
$C_u \leftarrow C_u \cup R$
Return K_R

EV(u, x)
If $u \neq U$: $y \leftarrow \mathsf{CF.Ev}^{\mathsf{P}}(1^\lambda, k_u, x)$
Else if $|x| < |X|$: $y \leftarrow ST_{|x|}$
Else if $|x| > |X|$: // $x \notin C_u$
  $y \leftarrow \Pi^{|x|-|X|-1}_{pub}(ST_{|X|-1})$
Else $y \leftarrow \bot$ // $x = X \notin C_u$
$S \leftarrow S \cup \{y\}$
If $x \notin C_u$:
  $z \leftarrow\!\!\$\ \mathcal{R}_\lambda$
  If $T_u[x] \neq \bot$: $z \leftarrow T_u[x]$
Else: $z \leftarrow$ PPRIM$(\mathsf{Ls}, (1, y), \bot)$
$E_u \leftarrow E_u \cup \{x\}$
$T_u[x] \leftarrow z$
Return z

Fig. 30. OW* adversary $\mathcal{A}_{(U,X)}$ for bounding the probability $\mathsf{bad}_{U,X}$ is set in G_1 (equivalently, G_4). We use that if $X = 1^\ell$, then $|X| = \ell$ and if $R = R_n = \{1^i : 0 \leq i \leq n\}$, then $|R| - 1 = n$.

2. $\Pr[\mathsf{G}_0(\lambda)] - \Pr[\mathsf{G}_1(\lambda)] \leq |N(\lambda)| \cdot \mathsf{Adv}^{\mathsf{ow}^*}_{\Pi,\mathsf{P},\mathcal{A}}(\lambda)$
4. $\Pr[\mathsf{G}_2(\lambda)] - \Pr[\mathsf{G}_3(\lambda)] = 0$

Let u denote an upper bound on how many different users $\mathcal{A}_{\mathsf{cprf}^*}$ will make queries for and l denote an upper bound on the longest input it will query to EV. Then $\mathcal{A}_{\mathsf{cprf}^*}$ will only make EV queries in $N(\lambda) = \{1, \ldots, u(\lambda)\} \times \{1^n : 0 \leq n \leq l(\lambda)\}$. (The restriction to user in $\{1, \ldots, u(\lambda)\}$ is without loss of generality by a simple renaming argument.)

The new fourth claim follows from the logic mentioned in the proof of Theorem 2 in Appendix D.1 that the difference between G_2 and G_3 would be 0 if P was consistent. For the new second claim, we define adversaries $\mathcal{A}_{U,X}$ such that $\Pr[\mathsf{G}_1 \text{ sets } \mathsf{bad}_{U,X}] \leq \mathsf{Adv}^{\mathsf{ow}^*}_{\Pi,\mathsf{P},\mathcal{A}_{U,X}}(\lambda)$ and then let $\mathcal{A}$ pick $(U, X) \leftarrow\!\!\$\ N(\lambda)$ and run $\mathcal{A}_{U,X}$.

Recall from the proof of Theorem 2 that G_1 will only set $\mathsf{bad}_{U,X}$ if (while X has not been constrained for U) $Y = \mathsf{CF.Ev}(k_U, X)$ is queried to the random oracle or internally computed inside of a EV query for inputs other than (U, X). If (tk, pub, ST_0) is the key of user U, then define $ST_i = \Pi^{-i}_{tk}(ST_0)$ for positive i (for one corner case we also need $ST_{-1} = \Pi_{pub}(ST_0)$). Note that because ST_0 is uniformly random and Π_{pub} is a permutation with inverse Π^{-1}_{tk}, sampling any of the ST_i at random and deriving the others from it gives an identical distribution. Consequently, our adversary will treat its input as $ST_{|X|-1}$. It stores the relevant y's queries to the random oracle or sampled inside EV in a set S, then at the end of the game or when $\mathcal{A}_{\mathsf{cprf}^*}$ makes a constrain query including (U, X) it checks if any of these are a Π_{pub} preimage of its input, winning if so. If $\mathsf{bad}_{U,X}$ would have been set, such a y must have been found.

Code of the adversary $\mathcal{A}_{U,X}$ is given in Fig. 30. For simplicity, we wrote it to match G_4 from Fig. 29 which we already argued is equivalent to G_1. For users other than U, it honestly executes the algorithms of CF. For U it precomputes ST_i for $i = |X|-1, \ldots, 0$. It uses these values as the output of $\mathsf{CF.Ev}$ or $\mathsf{CF.Con}$ whenever relevant. Consistency and non-programmability of P ensures this precomputing does not change the distribution of the game. Because we abort whenever X (and hence also anything longer than X) would be constrained for U, we never need any ST_i for $i \geq |X|$ to respond properly to $\mathcal{A}_{\mathsf{cprf}^*}$. However, we still want to capture if some (U, x) with $|x| > |X|$ is queried to EV which would compute a y equal to $ST_{|X|}$. If this was the came, then $ST_{|x|} = y = ST_{|X|}$ so we have found a cycle in Π_{pub}. Thus applying Π_{pub} to $ST_{|X|-1}$ the appropriate number of times ($|x|-|X|-1$) would recover $ST_{|X|}$. So we add $\Pi^{|x|-|X|-1}_{pub}(ST_{|X|-1})$ to S for this case. Adversary $\mathcal{A}_{(U,X)}$ correctly simulates the view of $\mathcal{A}_{\mathsf{cprf}^*}$ (up to when (U, X) would be constrained and hence $\mathsf{bad}_{U,X}$ could no longer be set) and wins its game whenever $\mathsf{bad}_{U,X}$ would occur. Hence $\Pr[\mathsf{G}_1 \text{ sets } \mathsf{bad}_{U,X}] \leq \mathsf{Adv}^{\mathsf{ow}^*}_{\Pi,\mathsf{P},\mathcal{A}_{U,X}}(\lambda)$ from which the second claim follows. □

E Proof of Theorem 4 (Security of Composed Construction)

We prove security of $\mathsf{CMP}[\mathsf{F},\mathsf{H}]$ security assuming security of the underlying constrained PRFs. In Appendix E.1, we prove IND*-NC-CPRF security. In Appendix E.2, we prove SIM*-AC-CPRF security.

E.1 Proof of IND*-NC-CPRF Security of CMP

The proof of IND*-NC-CPRF for $\mathsf{CMP}[\mathsf{F},\mathsf{H}]$ is fairly straightforward. We switch outputs of H, then F to random. The only subtlety is that an attacker can query (x', x'') to Ev and $R_{x',S}$ to Con as long as $x'' \notin S$. Then when the F key that H derives on input x' is switched to random in Ev, we must also use the appropriate random value in Con. If P was consistent, then the H adversary could simply reuse this key in Con. We are trying to avoid that assumption, so we instead use our convention that IND*-NC-CPRF allow Ev queries after Con queries as long as they are repeats of earlier Ev queries.

We construct a series of hybrid games G_0 to G_3 (Fig. 31), with G_0 being $\mathsf{G}^{\mathsf{ind^*\text{-}nc\text{-}cprf}}$ and G_3 being a game where the adversary's view is independent of the bit b. Hybrid G_0 is obtained by plugging in the code of $\mathsf{CMP}[\mathsf{F},\mathsf{H}]$, while G_1 does basic rewriting so that when a query $\textsc{Con}(u, R_{x',S})$ is made where no $\textsc{Ev}(u,(x',x''))$ query has been made, we replace evaluation of H on x' with constraining H on $\{x\}$. Then G_2 switches the output of $\mathsf{H.Ev}$ to random everywhere it is used (inside of Ev queries or $\textsc{Con}(u, R_{x',S})$ queries when a prior $\textsc{Ev}(u,(x',x''))$ query was made). Game G_3 switches the output of $\mathsf{F.Ev}$ to random in Ev. We define IND*-NC-CPRF adversaries $\mathcal{B}$ and $\mathcal{C}$ (Fig. 32) and establish the following claims.

1. $\Pr[\mathsf{G}_0(\lambda)] = \Pr[\mathsf{G}^{\mathsf{ind^*\text{-}nc\text{-}cprf}}_{\mathsf{CMP}[\mathsf{F},\mathsf{H}],\mathsf{P},\mathcal{A}_{\mathsf{cprf}}}(\lambda)]$
2. $\Pr[\mathsf{G}_1(\lambda)] = \Pr[\mathsf{G}_0(\lambda)]$
3. $\Pr[\mathsf{G}_2(\lambda)] - \Pr[\mathsf{G}_1(\lambda)] \leq \mathsf{Adv}^{\mathsf{ind^*\text{-}nc\text{-}cprf}}_{\mathsf{H},\mathsf{P},\mathcal{B}}(\lambda)$
4. $\Pr[\mathsf{G}_3(\lambda)] - \Pr[\mathsf{G}_2(\lambda)] \leq \mathsf{Adv}^{\mathsf{ind^*\text{-}nc\text{-}cprf}}_{\mathsf{F},\mathsf{P},\mathcal{C}}(\lambda)$
5. $\Pr[\mathsf{G}_3(\lambda) \leq 1/2$

Therefore,

$$\begin{aligned}
\mathsf{Adv}^{\mathsf{ind^*\text{-}nc\text{-}cprf}}_{\mathsf{CMP}[\mathsf{F},\mathsf{H}],\mathsf{P},\mathcal{A}_{\mathsf{cprf}}}(\lambda) &= 2\Pr[\mathsf{G}^{\mathsf{ind^*\text{-}nc\text{-}cprf}}_{\mathsf{CMP}[\mathsf{F},\mathsf{H}],\mathsf{P},\mathcal{A}_{\mathsf{cprf}}}(\lambda)] - 1 \\
&= 2\Pr[\mathsf{G}_1(\lambda)] - 1 \\
&= 2\Pr[\mathsf{G}_1(\lambda)] - 2\Pr[\mathsf{G}_2(\lambda)] + 2\Pr[\mathsf{G}_2(\lambda)] - 2\Pr[\mathsf{G}_3(\lambda)] + 2\Pr[\mathsf{G}_3(\lambda)] - 1 \\
&\leq 2\mathsf{Adv}^{\mathsf{ind^*\text{-}nc\text{-}cprf}}_{\mathsf{H},\mathsf{P},\mathcal{B}}(\lambda) + 2\mathsf{Adv}^{\mathsf{ind^*\text{-}nc\text{-}cprf}}_{\mathsf{F},\mathsf{P},\mathcal{C}}(\lambda)
\end{aligned}$$

Hybrid games G_2 and G_3 are identical when $b = 0$, so one can modify $\mathcal{C}$ and the analysis thereof to remove the factor 2 in front of its advantage.

Claim 1. Hybrid game G_0 why defined by simply plugging the code of $\mathsf{CMP}[\mathsf{F},\mathsf{H}]$ into $\mathsf{G}^{\mathsf{ind^*\text{-}nc\text{-}cprf}}$. This gives the first claim.

Claim 2. Hybrid game G_1 splits the behavior of Con on input sets of the form $R = R_{x',S}$. If x' was part of a prior query to Ev for the same user, then the behavior is as before. Otherwise, $\mathsf{H.Ev}^{\mathsf{P}}(1^\lambda, k_u, x')$ is replaced with $\mathsf{H.Con}^{\mathsf{P}}(1^\lambda, k_u, \{x'\})$. The second claim follows by our convention that the latter simply runs the former.

Claim 3. Hybrid game G_2 switches the output of $\mathsf{H.Ev}$ to being picked by a random function (both in Ev and Con). This is bounded by the adversary $\mathcal{B}$ which simulates G_1 or G_2 for $\mathcal{A}_{\mathsf{cprf}}$, replacing uses of H with oracle queries. Note that the Ev queries inside of its simulation of constrain (which may occur after it has already queried Con) can only be repeats of earlier Ev queries. Thus, it correctly simulates the view in either of the two games giving the third claim.

Games $\mathsf{G}_i(\lambda)$, $i \in \{0,\ldots,3\}$

$\sigma_{\mathsf{P}} \leftarrow\!\!\$\ \mathsf{P.Init}(1^\lambda)$
$k_{(\cdot)} \leftarrow\!\!\$\ \mathsf{H.K}(\lambda)$
$f^{\mathsf{H}}_{(\cdot)} \leftarrow\!\!\$\ \mathsf{H.Fcs}(\lambda)$ // $2 \le i$
$f^{\mathsf{F}}_{(\cdot,\cdot)} \leftarrow\!\!\$\ \mathsf{F.Fcs}(\lambda)$ // $3 \le i$
$f_{(\cdot)} \leftarrow\!\!\$\ \mathsf{CMP}[\mathsf{F},\mathsf{H}].\mathsf{Fcs}(\lambda)$
$b \leftarrow\!\!\$\ \{0,1\}$
$b' \leftarrow\!\!\$\ \mathcal{A}^{\textsc{Con},\textsc{Ev},\textsc{PPrim}}_{\mathsf{cprf}}(1^\lambda)$
Return $(b = b')$

$\textsc{PPrim}(\mathsf{Op}, x, y)$
Return $\mathsf{P.Op}(1^\lambda, x, y : \sigma_{\mathsf{P}})$

$\textsc{Con}(u, R)$
Require $R \cap E_u = \emptyset$
If $R = R_S$:
  $K_R \leftarrow \mathsf{H.Con}^{\mathsf{P}}(1^\lambda, k_u, S)$
Else if $R = R_{x',S}$:
  $K_{u,x'} \leftarrow \mathsf{H.Ev}^{\mathsf{P}}(1^\lambda, k_u, x')$ // $i < 1$
  $K_R \leftarrow \mathsf{F.Con}^{\mathsf{P}}(1^\lambda, K_{u,x'}, S)$ // $i < 1$
  If $\exists x''$ s.t. $(x', x'') \in E_u$: // $1 \le i$
    $K_{u,x'} \leftarrow \mathsf{H.Ev}^{\mathsf{P}}(1^\lambda, k_u, x')$ // $1 \le i < 2$
    $K_{u,x'} \leftarrow f^{\mathsf{H}}_u(x')$ // $2 \le i$
    $K_R \leftarrow \mathsf{F.Con}^{\mathsf{P}}(1^\lambda, K_{u,x'}, S)$ // $1 \le i$
  Else: // $1 \le i$
    $K_{u,x'} \leftarrow \mathsf{H.Con}^{\mathsf{P}}(1^\lambda, k_u, \{x'\})$ // $1 \le i$
    $K_R \leftarrow \mathsf{F.Con}^{\mathsf{P}}(1^\lambda, K_{u,x'}, S)$ // $1 \le i$
$C_u \leftarrow C_u \cup R$
Return K_R

$\textsc{Ev}(u, \boldsymbol{x})$
Require $\boldsymbol{x} \notin C_u$
Require $C_u = \emptyset$ or $\boldsymbol{x} \in E_u$
If $b = 1$:
  If $\boldsymbol{x} \in \mathsf{H.Inp}_1$:
    $y \leftarrow \mathsf{H.Ev}^{\mathsf{P}}(1^\lambda, k_u, \boldsymbol{x})$ // $i < 2$
    $y \leftarrow f^{\mathsf{H}}_u(\boldsymbol{x})$ // $2 \le i$
  Else if $\boldsymbol{x} = (x', x'') \in \mathsf{H.Inp}_2 \times \mathsf{F.Inp}$:
    $K_{u,x'} \leftarrow \mathsf{H.Ev}^{\mathsf{P}}(1^\lambda, k_u, x')$ // $i < 2$
    $K_{u,x'} \leftarrow f^{\mathsf{H}}_u(x')$ // $2 \le i < 3$
    $y \leftarrow \mathsf{F.Ev}^{\mathsf{P}}(1^\lambda, K_{u,x'}, x'')$ // $i < 3$
    $y \leftarrow f^{\mathsf{F}}_{u,x'}(x'')$ // $3 \le i$
If $b = 0$: $y \leftarrow f_u(\boldsymbol{x})$
$E_u \leftarrow E_u \cup \{\boldsymbol{x}\}$
Return y

Fig. 31. Hybrid games used for proving IND*-NC-CPRF security of $f_{(\cdot)} \leftarrow\!\!\$\ \mathsf{CMP}[\mathsf{F},\mathsf{H}].\mathsf{Fcs}(\lambda)$. Comments indicate which hybrids a line is included in. Note that for a transition into hybrid G_j, comments that end with j are added and comments that begin with j are removed.

Adversary $\mathcal{B}^{\textsc{Con},\textsc{Ev},\textsc{PPrim}}(1^\lambda)$

$f_{(\cdot)} \leftarrow\!\!\$\ \mathsf{CMP}[\mathsf{F},\mathsf{H}].\mathsf{Fcs}(\lambda)$
$b \leftarrow\!\!\$\ \{0,1\}$
$b' \leftarrow\!\!\$\ \mathcal{A}^{\textsc{ConSim},\textsc{EvSim},\textsc{PPrim}}_{\mathsf{cprf}}(1^\lambda)$
If $b = b'$: Return 1
Return 0

$\textsc{ConSim}(u, R)$
Require $R \cap E_u = \emptyset$
If $R = R_S$:
  $K_R \leftarrow \textsc{Con}(u, S)$
Else if $R = R_{x',S}$:
  If $\exists x''$ s.t. $(x', x'') \in E_u$:
    $K_{u,x'} \leftarrow \textsc{Ev}(u, x')$
    $K_R \leftarrow \mathsf{F.Con}^{\mathsf{P}}(1^\lambda, K_{u,x'}, S)$
  Else:
    $K_{u,x'} \leftarrow \textsc{Con}(u, \{x'\})$
    $K_R \leftarrow \mathsf{F.Con}^{\mathsf{P}}(1^\lambda, K_{u,x'}, S)$
$C_u \leftarrow C_u \cup R$
Return K_R

$\textsc{EvSim}(u, \boldsymbol{x})$
Require $\boldsymbol{x} \notin C_u$
Require $C_u = \emptyset$ or $\boldsymbol{x} \in E_u$
If $b = 1$:
  If $\boldsymbol{x} \in \mathsf{H.Inp}_1$:
    $y \leftarrow \textsc{Ev}(u, \boldsymbol{x})$
  Else if $\boldsymbol{x} = (x', x'') \in \mathsf{H.Inp}_2 \times \mathsf{F.Inp}$:
    $K_{x'} \leftarrow \textsc{Ev}(u, x')$
    $y \leftarrow \mathsf{F.Ev}^{\mathsf{P}}(1^\lambda, K_{u,x'}, x'')$
If $b = 0$: $y \leftarrow f_u(\boldsymbol{x})$
$E_u \leftarrow E_u \cup \{\boldsymbol{x}\}$
Return y

Adversary $\mathcal{C}^{\textsc{Con},\textsc{Ev},\textsc{PPrim}}(\lambda)$

$k_{(\cdot)} \leftarrow\!\!\$\ \mathsf{H.K}(\lambda)$
$f^{\mathsf{H}}_{(\cdot)} \leftarrow\!\!\$\ \mathsf{H.Fcs}(\lambda)$
$f_{(\cdot)} \leftarrow\!\!\$\ \mathsf{CMP}[\mathsf{F},\mathsf{H}].\mathsf{Fcs}(\lambda)$
$b \leftarrow\!\!\$\ \{0,1\}$
$b' \leftarrow\!\!\$\ \mathcal{A}^{\textsc{ConSim},\textsc{EvSim},\textsc{PPrim}}_{\mathsf{cprf}}(1^\lambda)$
If $b = b'$: Return 1
Return 0

$\textsc{ConSim}(u, R)$
Require $R \cap E_u = \emptyset$
If $R = R_S$:
  $K_R \leftarrow \mathsf{H.Con}^{\mathsf{P}}(1^\lambda, k_u, S)$
Else if $R = R_{x',S}$:
  If $\exists x''$ s.t. $(x', x'') \in E_u$:
    $K_R \leftarrow \textsc{Con}((u, x'), S)$
  Else:
    $K_{u,x'} \leftarrow \mathsf{H.Con}^{\mathsf{P}}(1^\lambda, k_u, \{x'\})$
    $K_R \leftarrow \mathsf{F.Con}^{\mathsf{P}}(1^\lambda, K_{u,x'}, S)$
$C_u \leftarrow C_u \cup R$
Return K_R

$\textsc{EvSim}(u, \boldsymbol{x})$
Require $\boldsymbol{x} \notin C_u$
Require $C_u = \emptyset$ or $\boldsymbol{x} \in E_u$
If $b = 1$:
  If $\boldsymbol{x} \in \mathsf{H.Inp}_1$:
    $y \leftarrow f^{\mathsf{H}}_u(\boldsymbol{x})$
  Else if $\boldsymbol{x} = (x', x'') \in \mathsf{H.Inp}_2 \times \mathsf{F.Inp}$:
    $y \leftarrow \textsc{Ev}((u, x'), x'')$
If $b = 0$: $y \leftarrow f_u(\boldsymbol{x})$
$E_u \leftarrow E_u \cup \{\boldsymbol{x}\}$
Return y

Fig. 32. IND*-NC-CPRF adversaries $\mathcal{B}$ (against H, transitioning from G_1 to G_2) and $\mathcal{C}$ (against F, transitioning from G_2 to G_3) used to prove security of $\mathsf{CMP}[\mathsf{F},\mathsf{H}]$.

Claim 4. Hybrid game G_3 switches the output of $\mathsf{F.Ev}$ to being picked by a random function in Ev. This is bounded by the adversary $\mathcal{C}$ which simulates G_2 or G_3 for $\mathcal{A}_{\mathsf{cprf}}$, replacing uses of F with oracle queries whenever F's keys are random from the function f^{H}. Importantly, the use of f^{H} that $\mathcal{C}$ still handles (for $\boldsymbol{x} \in \mathsf{H.Inp}_1$ in Ev) which never overlap with the uses it lets its oracles hand (for $x' \in \mathsf{H.Inp}_1$ in Ev or Con). Thus, it correctly simulates the view in either of the two games giving the fourth claim.

Claim 5. Finally, we note that the view of $\mathcal{A}_{\mathsf{cprf}}$ no longer depends on the bit b. Its only use is in Ev to decide whether the output comes from f or one of f^{H} and f^{F}. In either case, it is an output of a random function which will not be evaluated on the same inputs anywhere else. Hence, the final claim follows. □

E.2 Proof of SIM*-AC-CPRF Security of CMP

We conclude by proving the SIM*-AC-CPRF security of CMP. We follow a similar style to the IND*-NC-CPRF proof in Appendix E.1, replacing first the output H then F with randomness/simulation. However, while the conceptual core is the same there are some additionally challenges in the low-level details of this proof. In particular, we face a "race condition" between the simulators for F and H as discussed in the main body. In addressing these two challenges, we will use two tables: `CKey` to keep track of constrained keys that have already been returned to the adverasry and `Own` to keep track of which simulator "owns" (and thus will pick) pick the intermediate key corresponding to some $x \in \mathsf{H.Inp}_1$.

We construct a series of games G_0 to G_7, with G_0 being the real world of $\mathsf{G}^{\mathsf{sim^*\text{-}ac\text{-}cprf}}$ and G_7 being the simulated world of $\mathsf{G}^{\mathsf{sim^*\text{-}ac\text{-}cprf}}$. We also define the corresponding simulator S in Fig. 35. Of these, we establish a series of claims:

1. $\Pr[\mathsf{G}_0(\lambda)] = \Pr[\mathsf{G}^{\mathsf{sim^*\text{-}ac\text{-}cprf}}_{\mathsf{CMP[F,H]},\mathsf{S},\mathsf{P},\mathcal{A}} | b = 1]$
2. $\Pr[\mathsf{G}_1(\lambda)] = \Pr[\mathsf{G}_0(\lambda)]$
3. $\Pr[\mathsf{G}_2(\lambda)] = \Pr[\mathsf{G}_1(\lambda)]$
4. $\Pr[\mathsf{G}_3(\lambda)] - \Pr[\mathsf{G}_2(\lambda)] = \mathsf{Adv}^{\mathsf{sim^*\text{-}ac\text{-}cprf}}_{\mathsf{H},\mathsf{S_H},\mathsf{P},\mathcal{B}}(\lambda)$
5. $\Pr[\mathsf{G}_4(\lambda)] = \Pr[\mathsf{G}_3(\lambda)]$
6. $\Pr[\mathsf{G}_5(\lambda)] = \Pr[\mathsf{G}_4(\lambda)]$
7. $\Pr[\mathsf{G}_6(\lambda)] - \Pr[\mathsf{G}_5(\lambda)] = \mathsf{Adv}^{\mathsf{sim^*\text{-}ac\text{-}cprf}}_{\mathsf{F},\mathsf{S_F},\mathsf{P},\mathcal{C}}(\lambda)$
8. $\Pr[\mathsf{G}_7(\lambda)] = \Pr[\mathsf{G}_6(\lambda)]$
9. $\Pr[\mathsf{G}_7(\lambda)] = \Pr[\neg\mathsf{G}^{\mathsf{sim^*\text{-}ac\text{-}cprf}}_{\mathsf{CMP[F,H]},\mathsf{S},\mathsf{P},\mathcal{A}} | b = 0]$

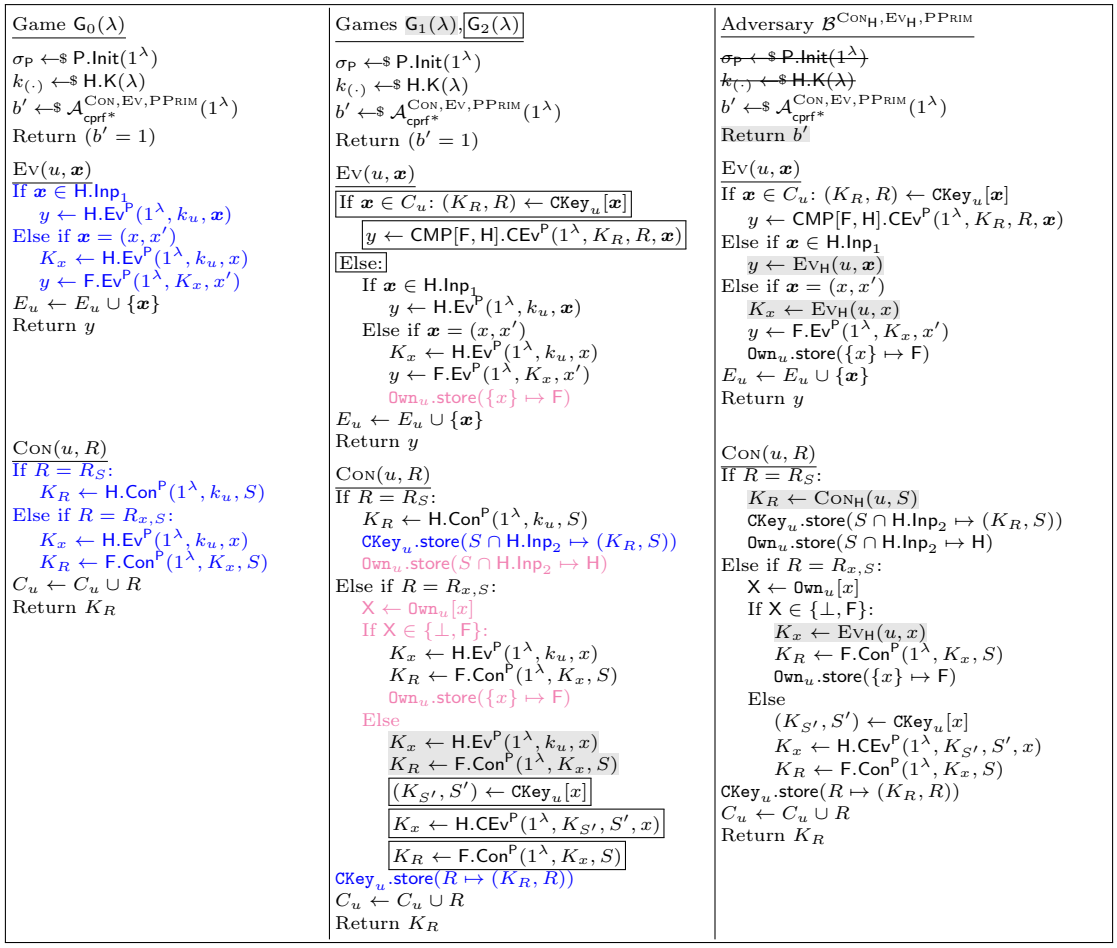
Therefore,

$$\begin{aligned}\mathsf{Adv}^{\mathsf{sim^*\text{-}ac\text{-}cprf}}_{\mathsf{CMP[F,H]},\mathsf{S},\mathsf{P},\mathcal{A}}(\lambda) &= \Pr[\mathsf{G}_0(\lambda)] - \Pr[\mathsf{G}_7(\lambda)] = \sum_{i=0}^{6} (\Pr[\mathsf{G}_i(\lambda)] - \Pr[\mathsf{G}_{i+1}(\lambda)]) \\ &= \mathsf{Adv}^{\mathsf{sim^*\text{-}ac\text{-}cprf}}_{\mathsf{H},\mathsf{S_H},\mathsf{P},\mathcal{B}}(\lambda) + \mathsf{Adv}^{\mathsf{sim^*\text{-}ac\text{-}cprf}}_{\mathsf{F},\mathsf{S_F},\mathsf{P},\mathcal{C}}(\lambda).\end{aligned}$$

The result follows from the assumed security of H and F.

Throughout this section, we write $T.\mathsf{store}(S \mapsto y)$ to mean an operation equivalent to "For $x \in S$: $T[x] \leftarrow y$". In case S is large, T will need to somehow represent this implicitly for efficiency.

Claims 1 and 2. Hybrid G_0 is defined on the top left of Fig. 33 is the real world ($b = 1$) of SIM*-AC-CPRF game defined in Fig. 2 with code of CMP plugged in. Hybrid G_1 defined on the top middle of Fig. 33 contains highlighted code. In this hybrid, we are merely adding tracking tables `CKey` and `Own` which are marked in blue and pink respectively. When a (x, x') evaluation query is made, F is assigned ownership of x (associated with the appropriate user u). Similarly when a key is constrained on $R_{x,S}$, if x is not already owned by F. When a R_S query is made, H takes ownership over all $x \in S \cap \mathsf{H.Inp}_2$. This may result in H taking over ownership of x's previously owned by F (in later games $\mathsf{S_F.Con}$ will be run for user (u, x) on set $R = \mathsf{F.Inp}$ here to derive a key for x that is then passed to $\mathsf{S_H.Con}$). Note that F will never take ownership of a value previously owned by H.



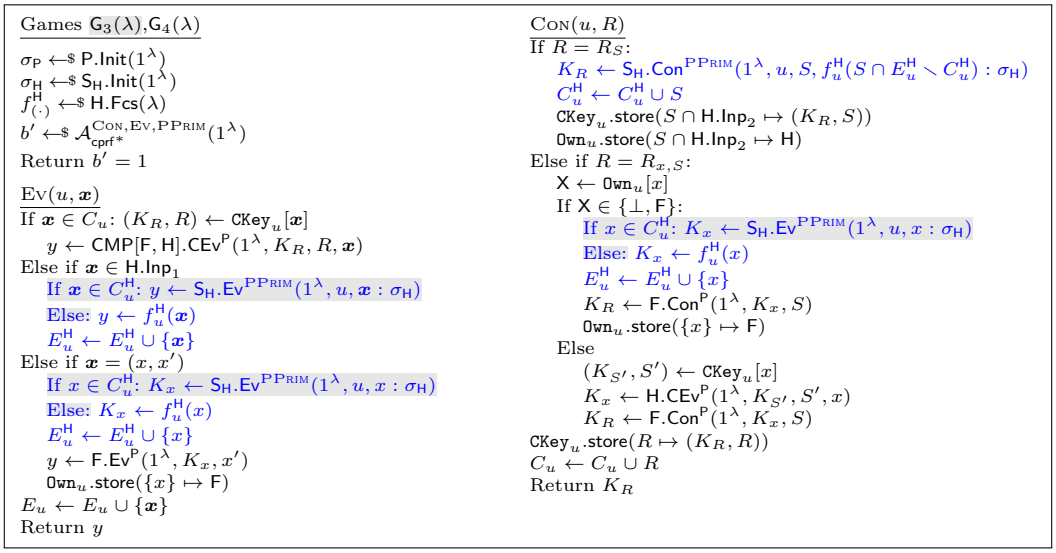


Fig. 33. Games and adversary for CMP security proof. G_0 plugs the code of CMP into the real world. G_1 adds code to track constrained keys and key owners. G_2 reuses constrained keys when possible. $\mathcal{B}$ reduces to security of H so that G_3 can replace it with simulation. G_4 removes an unused branches.



Fig. 34. Games and adversary for CMP security proof. G_5 uses keys in place of f^{H} for x's "owned" by F, but re-inserts then into f^{H} when ownership changes. $\mathcal{C}$ reduces to security of F so that G_6 can replace it with $\boxed{\text{simulation}}$. ~~Struck code~~ in G_6 are unused branches.

Claim 3. Hybrid G_2 is defined on the top middle of Fig. 34. It contains boxed code but not highlighted code. In this hybrid, we make use of constrained keys stored in $\mathtt{CKey}$ to respond to some evaluation queryies. Notice that we always update $\mathtt{CKey}$ with the constrained keys for CMP at the end of Con. Consequently, $\mathtt{CKey}$ will be non-$\perp$ for any inputs in C_u. So these keys are used in Ev whenever the input is in C to do constrained evaluation instead of evaluation.

We additionally update $\mathtt{CKey}$ inside Con in the "If $R = R_S$" branch to store constrained keys for H. Note that H is also given ownership of all inputs that these keys are being stored for. These constrained keys allow for evaluating inputs to CMP in the form of $(S \cap \mathsf{H.Inp}_2\mathsf{H}) \times \mathsf{F.Inp}$. Then later when a Con query is made in the form of $\{x\} \times \mathsf{F.Inp}$ where $x \in \mathsf{H.Inp}_2$ is owned by H, we replace H evaluation with constrained evaluation using the stored key.

The third claim follows from the strong consistency of H and F.

Claim 4. Hybrid G_3 is defined on the bottom of Fig. 33. It contains highlighted code. In this hybrid, we want to replace the output of H with random or simulated values. We additionally keeps sets E_u^{H} and C_u^{H} set to match the corresponding sets in the coming reduction adversary's game against H. SIM*-AC-CPRF adversary $\mathcal{B}$ for H is defined on the top right of Fig. 33 to bound the difference between $\Pr[\mathsf{G}_2(\lambda)]$ and $\Pr[\mathsf{G}_3(\lambda)]$. The highlighted code is where $\mathcal{B}$ uses its own oracles to perform evaluations of H.

Let b_{H} be the hidden bit in $\mathcal{B}$'s game. When b_{H} is 1, $\mathcal{B}$ is emulating G_2 for $\mathcal{A}$. When b_{H} is 0, $\mathcal{B}$ is emulating G_3 for $\mathcal{A}$. The correctness of this emulation gives the fourth claim.

Claims 5 and 6. Hybrid G_4 is defined on the bottom of Fig. 33 and does not contain highlighted code. It removes unnecessary branches in G_3. Notice whenever we update C_u^{H} with set S in Con, we always update $\mathtt{Own}_u[x]$ to H for any $x \in S \cap \mathsf{H.Inp}_2$ and update C_u with R_S. Therefore, inside Ev, in the "$\boldsymbol{x} \notin C_u$" branch, we know $\boldsymbol{x} \notin C_u^{\mathsf{H}}$ will hold. Inside Con, when $\mathtt{Own}_u[x] \neq \mathsf{H}$, we know $\boldsymbol{x} \notin C_u^{\mathsf{H}}$ will hold. So the branches are indeed unused and can be removed.

Hybrid G_5 is defined on the left of Fig. 34 and contains highlighted code. In G_4 there are several places where keys for F which are owned by F were chosen by the random function f^{H}. Now in G_5, we instead lazily sample such keys using k. Inside of Con when $R = R_S$, the simulator S_{H} expects be given the partial function of f^{H} restricted to $S \cap E_u^{\mathsf{H}} \setminus C_u^{\mathsf{H}}$. We create this partial function by looping over all x's in S that are owned by F and inserting their keys into f^{H}. Since both k and f^{H} are both sampled uniformly at random and otherwise unused, G_5 is identical to G_4. This gives the sixth claim.

Claim 7. Hybrid G_6 is defined on the left of Fig. 34 and contains boxed but not highlighted code. We are switching outputs of F to be random or simulated. Similarly, we keep track of C^{F} and E^{F} to match the sets in the reduction adversary's game. The struck code of G_6 is unreachable because $x' \in C_{u,x}^{\mathsf{F}}$ implies $\boldsymbol{x} \in C_u$. SIM*-AC-CPRF adversary $\mathcal{C}$ for F is defined on the right of Fig. 34 to bound the difference between $\Pr[\mathsf{G}_5(\lambda)]$ and $\Pr[\mathsf{G}_6(\lambda)]$. The highlighted code is where $\mathcal{B}$ replaces evaluations of F using its own oracles to emulate the game for $\mathcal{A}$.

Let b_{F} be the hidden bit in$\mathcal{C}$'s game, when b_{F} is 1, $\mathcal{C}$ is emulating G_5 for $\mathcal{A}$. When b_{F} is 0, $\mathcal{C}$ is emulating G_6 for $\mathcal{A}$. Hence claim 7 follows.

Claims 8 and 9. Hybrid G_7 is defined on the left of Fig. 35. Now, instead of sampling f^{H} and f^{F} in the beginning, we only sample a single f for CMP. Later, we temporarily reconstruct the appropriate f^{H} and f^{F} using f inside Con. Importantly, F owns a relevant x, we still program f^{H} using values from S_{F}. This change does not modify the behavior of the game and so the eighth claim holds.

Finally, we define the simulator on the right of Fig. 35. It was designed so that G_7 matches the simulated world of the SIM*-AC-CPRF game for $\mathsf{CMP}[\mathsf{F}, \mathsf{H}]$. We emphasize that depends on the simulators for F and H, but not on the given adversary. This justifies claim 9 and hence completes the proof. □

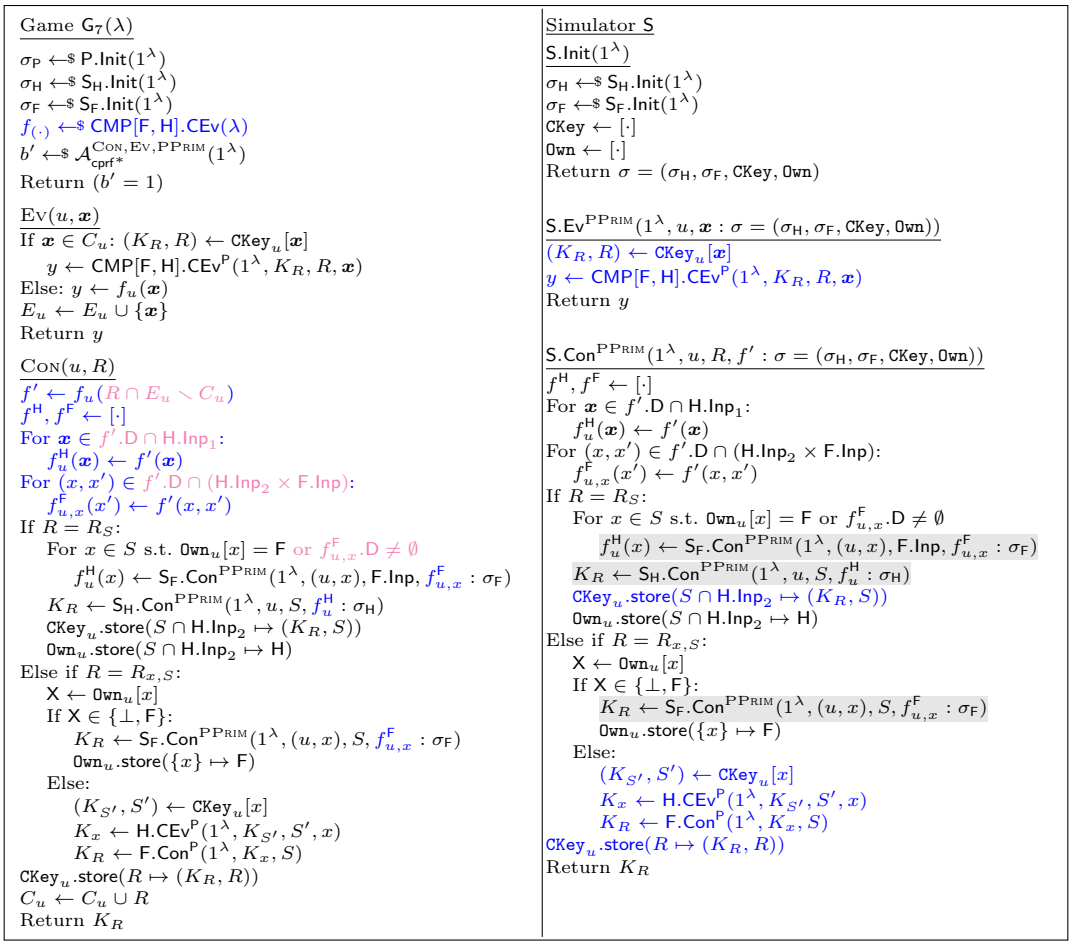


Fig. 35. Games and adversary for CMP security proof. G_7 uses a combined f in place of f^{H} and f^{F}. When needed, it defines components of f^{H} and f^{F} from f. It stops tracking when Own is used in Ev and sets E^{H}, C^{H}, E^{F}, and C^{F}. It infers the needed information from E and C. Simulator S perfectly simulates G_7 by reusing constrained keys and simulating H and F.

F Proof for Theorem 5 (Security of Cascade-based Constructions)

Proof. The proof divides into two parts. First we argue syntactically why both CSC and ACSC can be viewed as being obtained from repeated applications of CMP using the underlying PRF. Then we give a generic argument using the proof of Theorem 4 to argue why schemes obtained by repeated application of CMP are secure.

To start, assume that the CPRF attacker under consideration will make queries of length at most $n(\lambda)$. That is, for any $\boldsymbol{x}$ queried to Ev and any $R_{\boldsymbol{x}\sqsubseteq}$ queried to Con, $\boldsymbol{x} \in \mathsf{F}.\mathsf{Inp}^{\leq n(\lambda)}$. For now, treat the security parameter λ as fixed.

Cascade Construction. We start with $\mathsf{CSC}[\mathsf{F},\mathsf{D}]$. It suffices to restrict the domain to inputs of length at most $n(\lambda)$ and to add inputs to ensure that every string is a prefix of or is prefixed by a string in the domain. Let $\mathsf{E} = \mathsf{D}_{\leq n(\lambda)} \cup \{\boldsymbol{x} \in \mathsf{F}.\mathsf{Inp}^{n(\lambda)} \mid \forall \boldsymbol{y} \in \mathsf{D}_{\leq n(\lambda)}, \boldsymbol{y} \not\sqsubseteq \boldsymbol{x}\}$ where we let $\mathsf{D}_{\leq n(\lambda)} = \mathsf{D} \cap \boldsymbol{x} \in \mathsf{F}.\mathsf{Inp}^{\leq n(\lambda)}$. (Using E in place of D adds minor notational convenience to the following.)

Now we recursively define a sequence of constrained function families CSC_i by $\mathsf{CSC}_1 = \mathsf{F}$ (viewing F as a constrained function family) and $\mathsf{CSC}_{i+1} = \mathsf{CMP}[\mathsf{F}, \mathsf{CSC}_i]$. For this use of CMP we use $\mathsf{CSC_i.Inp}_1 = \mathsf{F.Inp}^{\leq i} \cap \mathsf{E}$ and $\mathsf{CSC_i.Inp}_2 = \mathsf{F.Inp}^i \cap \{\boldsymbol{x} \mid \exists y \in \mathsf{E}, \boldsymbol{x} \sqsubset y\}$.

Our claim is that from the perspective of an attacker making queries of length at most $n(\lambda)$, $\mathsf{CSC}_{n(\lambda)}$ and $\mathsf{CSC}[\mathsf{F}, \mathsf{D}]$ are equivalent.[7] Formally this follows from an induction, the details of which are straightforward.

Augmented Cascade. Next we consider $\mathsf{ACSC}[\mathsf{Chop}, \mathsf{F}]$. Recursively define $\mathsf{ACSC}_0 = \mathsf{F}_{\mathsf{Chop}}$ and $\mathsf{ACSC}_{i+1} = \mathsf{CMP}[\mathsf{F}_{\mathsf{Chop}}, \mathsf{ACSC}_i]$. For this use of CMP we set $\mathsf{ACSC_i.Inp}_1 = \mathsf{F.Inp}^{\leq i}$ and $\mathsf{ACSC_i.Inp}_2 = \mathsf{F.Inp}^{i+1}$. Note that (identifying $\mathsf{F.Inp}^{i+1} \times \{\varepsilon\}$ with $\mathsf{F.Inp}^{i+1}$) this gives us

$$\mathsf{ACSC_{i+1}.Inp} = \mathsf{F.Inp}^{\leq i} \sqcup \mathsf{F.Inp}^{i+1} \times (\{\varepsilon\} \sqcup \mathsf{F.Inp}) = \mathsf{F.Inp}^{\leq i+2}.$$

Our claim is that from the perspective of an attacker making queries of length at most $n(\lambda)$, $\mathsf{ACSC}_{n(\lambda)}$ and $\mathsf{ACSC}[\mathsf{Chop}, \mathsf{F}]$ are equivalent. This holds because

$$\mathsf{ACSC_i.Ev^P}(1^\lambda, k, \boldsymbol{x}) = \begin{cases} \mathsf{ACSC}[\mathsf{Chop}, \mathsf{F}].\mathsf{Ev^P}(1^\lambda, k, \boldsymbol{x}), & \boldsymbol{x} \in \mathsf{F.Inp}^{\leq i} \\ \mathsf{CSC}[\mathsf{F}].\mathsf{Ev^P}(1^\lambda, k, \boldsymbol{x}), & \boldsymbol{x} \in \mathsf{F.Inp}^{i+1}. \end{cases}$$

Formally this follows from an induction, the details of which are straightforward.

Security Argument. From the security of Theorem 4 showing the SIM*-AC-CPRF security of $\mathsf{CMP}[\mathsf{F}, \mathsf{H}]$ (given in Appendix E.2), we get three efficient maps $M_1 : (\mathsf{F}, \mathsf{H}, \mathcal{A}_{\mathsf{cprf}^*}) \mapsto \mathcal{B}$, $M_2 : (\mathsf{F}, \mathsf{H}, \mathcal{A}_{\mathsf{cprf}^*}, \mathsf{S_H}) \mapsto \mathcal{C}$, and $M_3 : (\mathsf{F}, \mathsf{H}, \mathsf{S_F}, \mathsf{S_H}) \mapsto \mathsf{S}$. Here $\mathcal{A}_{\mathsf{cprf}^*}$ is the given CMP adversary, $\mathsf{S_F}$ is the given F simulator, and $\mathsf{S_H}$ is the given H simulator. The maps correspond to how the proof used these to construct adversary $\mathcal{B}$ for H, adversary $\mathcal{C}$ for F, and simulator S for CMP. The proof establishes that

$$\mathsf{Adv}^{\mathsf{sim^*\text{-}ac\text{-}cprf}}_{\mathsf{CMP}[\mathsf{F},\mathsf{H}],M_3(\mathsf{F},\mathsf{H},\mathsf{S_F},\mathsf{S_H}),\mathsf{P},\mathcal{A}_{\mathsf{cprf}^*}}(\lambda) = \mathsf{Adv}^{\mathsf{sim^*\text{-}ac\text{-}cprf}}_{\mathsf{H},\mathsf{S_H},\mathsf{P},M_1(\mathsf{F},\mathsf{H},\mathcal{A}_{\mathsf{cprf}^*})}(\lambda) + \mathsf{Adv}^{\mathsf{sim^*\text{-}ac\text{-}cprf}}_{\mathsf{F},\mathsf{S_F},\mathsf{P},M_2(\mathsf{F},\mathsf{H},\mathcal{A}_{\mathsf{cprf}^*},\mathsf{S_H})}(\lambda).$$

holds for all $\mathsf{F}, \mathsf{H}, \mathcal{A}_{\mathsf{cprf}^*}, \mathsf{S_F}, \mathsf{S_H}$.

Now we apply this for proving CSC secure. Let $\mathcal{A}_{\mathsf{CSC}}$ be the given adversary for CSC making queries of length at most $n(\lambda)$ and $\mathsf{S_F}$ be the given simulator for F. Recursively define $\mathsf{S}_1 = \mathsf{S_F}$ and $\mathsf{S}_{i+1} = M_3(\mathsf{F}, \mathsf{CSC}_i, \mathsf{S_F}, \mathsf{S}_i)$ for $i = 1, \ldots, n(\lambda) - 1$. Recursively define $\mathcal{B}_{n(\lambda)} = \mathcal{A}_{\mathsf{CSC}}$ and $\mathcal{B}_{i-1} = M_1(\mathsf{F}, \mathsf{CSC}_{i-1}, \mathcal{B}_i)$ for $i = n(\lambda), \ldots, 2$. Define $\mathcal{C}_{i-1} = M_2(\mathsf{F}, \mathsf{CSC}_{i-1}, \mathcal{B}_i, \mathsf{S}_{i-1})$ for $i = n(\lambda), \ldots, 2$. Then for $i = 1, \ldots, n(\lambda) - 1$ we have,

$$\begin{aligned} \mathsf{Adv}^{\mathsf{sim^*\text{-}ac\text{-}cprf}}_{\mathsf{CSC}_{i+1},\mathsf{S}_{i+1},\mathsf{P},\mathcal{B}_{i+1}}(\lambda) &= \mathsf{Adv}^{\mathsf{sim^*\text{-}ac\text{-}cprf}}_{\mathsf{CSC}_{i+1},M_3(\mathsf{F},\mathsf{CSC}_i,\mathsf{S_F},\mathsf{S}_i),\mathsf{P},\mathcal{B}_{i+1}}(\lambda) \\ &= \mathsf{Adv}^{\mathsf{sim^*\text{-}ac\text{-}cprf}}_{\mathsf{CMP}[\mathsf{F},\mathsf{CSC}_i],M_3(\mathsf{F},\mathsf{CSC}_i,\mathsf{S_F},\mathsf{S}_i),\mathsf{P},\mathcal{B}_{i+1}}(\lambda) \\ &= \mathsf{Adv}^{\mathsf{sim^*\text{-}ac\text{-}cprf}}_{\mathsf{CSC}_i,\mathsf{S}_i,\mathsf{P},M_1(\mathsf{F},\mathsf{CSC}_i,\mathcal{B}_{i+1})}(\lambda) + \mathsf{Adv}^{\mathsf{sim^*\text{-}ac\text{-}cprf}}_{\mathsf{F},\mathsf{S_F},\mathsf{P},M_2(\mathsf{F},\mathsf{CSC}_i,\mathcal{B}_{i+1},\mathsf{S}_i)}(\lambda). \\ &= \mathsf{Adv}^{\mathsf{sim^*\text{-}ac\text{-}cprf}}_{\mathsf{CSC}_i,\mathsf{S}_i,\mathsf{P},\mathcal{B}_i}(\lambda) + \mathsf{Adv}^{\mathsf{sim^*\text{-}ac\text{-}cprf}}_{\mathsf{F},\mathsf{S_F},\mathsf{P},\mathcal{C}_i}(\lambda). \end{aligned}$$

Recall that the view of $\mathcal{A}_{\mathsf{CSC}}$ interacting with $\mathsf{CSC}[\mathsf{F}, \mathsf{D}]$ is identical to its view interacting with $\mathsf{CSC}_{n(\lambda)}$. So repeated applications of the above gives

$$\mathsf{Adv}^{\mathsf{sim^*\text{-}ac\text{-}cprf}}_{\mathsf{CSC}[\mathsf{F},\mathsf{D}],\mathsf{S}_{n(\lambda)},\mathsf{P},\mathcal{A}_{\mathsf{CSC}}}(\lambda) = \mathsf{Adv}^{\mathsf{sim^*\text{-}ac\text{-}cprf}}_{\mathsf{F},\mathsf{S_F},\mathsf{P},\mathcal{B}_1}(\lambda) + \sum_{i=1}^{n(\lambda)-1} \mathsf{Adv}^{\mathsf{sim^*\text{-}ac\text{-}cprf}}_{\mathsf{F},\mathsf{S_F},\mathsf{P},\mathcal{C}_i}(\lambda).$$

Let $\mathcal{D}$ be an adversary that randomly selects an adversary from $\{\mathcal{B}_1, \mathcal{C}_1, \ldots, \mathcal{C}_{n(\lambda)-1}\}$ then runs that adversary we get

$$\mathsf{Adv}^{\mathsf{sim^*\text{-}ac\text{-}cprf}}_{\mathsf{CSC}[\mathsf{F},\mathsf{D}],\mathsf{S}_{n(\lambda)},\mathsf{P},\mathcal{A}_{\mathsf{CSC}}}(\lambda) = n(\lambda) \cdot \mathsf{Adv}^{\mathsf{sim^*\text{-}ac\text{-}cprf}}_{\mathsf{F},\mathsf{S_F},\mathsf{P},\mathcal{D}}(\lambda).$$

[7] Technically, this does not cover the case when $\varepsilon \in \mathsf{D}$, but then $\mathsf{D} = \{\varepsilon\}$ holds. Security of $\mathsf{CSC}[\mathsf{F}, \{\varepsilon\}]$ is trivial.

This almost concludes the proof, except that $\mathsf{S}_{n(\lambda)}$ depends on n and thus $\mathcal{A}_{\mathsf{CSC}}$. We need a single S defined such that for any $\mathcal{A}_{\mathsf{CSC}}$ making queries of length at most $n(\lambda)$ will end up behaving identically to $\mathsf{S}_{n(\lambda)}$.

The basic idea is for S to run like S_l where l is the longest query it's seen so far (whether from $\boldsymbol{x}$ given to $\mathsf{S.Ev}$, $R_{\boldsymbol{x}\sqsubset}$ given to $\mathsf{S.Con}$, or $\boldsymbol{x}$ queried to Ev and then later included as part of the domain of an f' given to $\mathsf{S.Con}$). It's not hard to verify that such behavior can be done consistently based on the simulator of Fig. 35 by noting that running S_{i+1} on inputs of length at most i essentially just makes S_i respond to all queries.

The details work identically for ACSC by replacing instances of CSC and F with ACSC and $\mathsf{F}_{\mathsf{Chop}}$ and defining S_i, $\mathcal{B}_i$, and $\mathcal{C}_i$ for $i = 0$ as well. We get

$$\mathsf{Adv}^{\mathsf{sim^*\text{-}ac\text{-}cprf}}_{\mathsf{ACSC[Chop,F]},\mathsf{S},\mathsf{P},\mathcal{A}_{\mathsf{CSC}}}(\lambda) = (n(\lambda)+1) \cdot \mathsf{Adv}^{\mathsf{sim^*\text{-}ac\text{-}cprf}}_{\mathsf{F}_{\mathsf{Chop}},\mathsf{S}_{\mathsf{F}_{\mathsf{Chop}}},\mathsf{P},\mathcal{D}}(\lambda).$$

where $\mathcal{D}$ selects an adversary from $\{\mathcal{B}_0, \mathcal{C}_0, \ldots, \mathcal{C}_{n(\lambda)-1}\}$ to run.

The security for IND*-NC security proceeds analogouly, except without the need for a simulator. □

G SIM*-AC-PRF Security With Leakage

In this section we analyze the security of low-level compression functions as SIM*-AC-PRFs in the face of key leakage. We start with some combinatorial preliminaries and then consider random oracles in Section G.1 and the Davies-Meyer construction in Section G.2.

Combinatorial Preliminaries. Output function Chop is a key-less function that maps from $\mathsf{Chop.Inp}(\lambda)$ to $\mathsf{Chop.Out}(\lambda)$. Let $\rho_{\mathsf{Chop}}(\lambda)$ be a lower bound on the number of preimages that any $y \in \mathsf{Chop.Out}(\lambda)$ has. We will require that this be large (superpolynomial). We require that the output of Chop on random input be statistically close to uniform, meaning

$$\delta_{\mathsf{Chop}}(\lambda) = \max_{\mathcal{A}} \Pr[1 \leftarrow\!\$\ \mathcal{A}(1^\lambda, \mathsf{Chop}(1^\lambda, x))] - \Pr[1 \leftarrow\!\$\ \mathcal{A}(1^\lambda, y)]$$

is negligible where $x \leftarrow\!\$\ \mathsf{Chop.Inp}(\lambda)$ and $y \leftarrow\!\$\ \mathsf{Chop.Out}(\lambda)$. Next we require that when sampling many independent $y = \mathsf{Chop}(1^\lambda, x)$, it is unlikely for any of these outputs to repeat many times. To formalize this, define $\mu(y_1, \ldots, y_u) = \max_y |\{i \ : \ y_i = y\}|$ and

$$\mathsf{P}^{\mathsf{coll}}_{\mathsf{Chop}}(\lambda, u, m) = \Pr[\mu(\mathsf{Chop}(1^\lambda, x_1), \ldots, \mathsf{Chop}(1^\lambda, x_u)) \geq m(\lambda)]$$

where $x_1, \ldots, x_u \leftarrow\!\$\ \mathsf{Chop.Inp}(\lambda)$. Bellare, et al. [5, Lemma 7.1] bound this as follows.

Lemma 1 (Multi-collision Probability). *Let $u \geq 1, \kappa \geq 0$ and $m \geq \kappa \cdot \ln|\mathsf{Chop.Out}| + 2u/|\mathsf{Chop.Out}|$. Then $\mathsf{P}^{\mathsf{coll}}_{\mathsf{Chop}}(u, m) \leq u \cdot \delta_{\mathsf{Chop}} + \exp(-\kappa/3)$.*

The analysis of AMAC as a PRF uses the above quantities [5]. We require an additional property of Chop, that it is possible to sample pre-images uniformly. This property is required so that simulators can find preimages to the previously random outputs of evaluation. For similar reasons, this property is also required when analyzing AMAC's indifferentiability from a random oracle [7]. Formally, we require the existence of a randomized algorithm Chop^{-1} such that $\mathsf{Chop}(1^\lambda, \mathsf{Chop}^{-1}(1^\lambda, y)) = y$ and

$$\varepsilon_{\mathsf{Chop}}(\lambda) = \max_{\mathcal{A}} \Pr[1 \leftarrow\!\$\ \mathcal{A}(1^\lambda, x)] - \Pr[1 \leftarrow\!\$\ \mathcal{A}(1^\lambda, \mathsf{Chop}^{-1}(1^\lambda, y))]$$

is negligible where $x \leftarrow\!\$\ \mathsf{Chop.Inp}(\lambda)$ and $y \leftarrow\!\$\ \mathsf{Chop.Out}(\lambda)$. Note that $\delta_{\mathsf{Chop}}(\lambda) \leq \varepsilon_{\mathsf{Chop}}(\lambda)$. Concrete bounds for these terms are known [5,7] for when Chop truncates a large bitstring or reduces a number modulo some fixed M.

G.1 Security of Ideal Compression Function (Random Oracle) with Leakage

We consider the case that our underlying compression function is a random oracle. Formally, we capture this by the function family which simply returns the output of its ideal primitive on (K, x). Recall that our leakage resilient definition in Section 3.1 was equivalent to considering plain SIM*-AC-PRF security of a related function family that used $\mathsf{Chop}(K)$ as the output on a new domain point.

Theorem 8. *Let* F *be a function family with* $\mathsf{F.Ev}^{\mathsf{P_{rom}}}(1^\lambda, K, x) = \mathsf{P_{rom}}(K, x)$ *and* $\varepsilon \notin \mathsf{F.Inp}$. *Let* $\mathsf{Chop}(1^\lambda, \cdot) : \mathsf{F.K}(\lambda) \to \mathsf{Chop.Out}(\lambda)$ *and* $m(\lambda) \geq 1$. *Let* $\mathsf{F_{Chop}}$ *be defined equivalently to* F, *except that* $\mathsf{F_{Chop}.Ev}^{\mathsf{P}}(1^\lambda, K, \varepsilon) = \mathsf{Chop}(1^\lambda, K)$. *Then we can build an efficient* S *such that*

$$\mathsf{Adv}^{\mathsf{sim^*\text{-}ac\text{-}prf}}_{\mathsf{F_{Chop}},\mathsf{S},\mathsf{P},\mathcal{A}_{\mathsf{prf}}}(\lambda) \leq \frac{u(\lambda)^2}{2|\mathsf{F_{Chop}.K}(\lambda)|} + \mathsf{P}^{\mathsf{coll}}_{\mathsf{Chop}}(\lambda, u(\lambda), m(\lambda)) + \frac{(m(\lambda)-1)p(\lambda)}{\rho_{\mathsf{Chop}}(\lambda)} + u(\lambda) \cdot \varepsilon_{\mathsf{Chop}}(\lambda)$$

holds for all $\mathcal{A}_{\mathsf{prf}}$ *making* $p(\lambda)$ *queries to* PPRIM, *and querying at most* $u(\lambda)$ *distinct users.*

This bound exactly matches the bound in Bellare, et al.'s analogous result in their Theorem 7.2, except we have an additional $u \cdot \varepsilon$. It does not arise in their proof because their definition of PRF security under leakage returns $\mathsf{Chop}(1^\lambda, K)$ in both the real and random world. If a random value was returned in the random world instead, it would add a $u \cdot \delta$ term to the bound. They don't need ε because they don't need a simulator applying Chop^{-1} to the random value. With their proof modularization, $\mathsf{Chop}(1^\lambda, K)$ looking random instead arises in the final step of analyzing ACSC or AMAC and arises in their Theorem 8.2 and Theorem 8.5 as a $q_{\mathrm{Ev}} \cdot \delta$ term where q_{Ev} is the number of evaluation queries made. With our modularization, combining Theorem 8 with Theorem 5 would give a $(q_{\mathrm{Ev}} + q_{\mathrm{Con}}) \cdot n \cdot \varepsilon$ term where q_{Con} is the number of constrained queries made and n is the length of the longest query to ACSC. This is small and so not important to optimize, though the "true" bound should be $q_{\mathrm{Ev}} \cdot \varepsilon$ because $\mathsf{Chop}(1^\lambda, \cdot)$ is never applied by constraining and is applied once per evaluation.

Proof. We define our simulator S as follows.

$\underline{\mathsf{S.Init}(1^\lambda)}$
$K_{(\cdot)} \leftarrow [\cdot]$
Return $K_{(\cdot)}$

$\underline{\mathsf{S.Ev}^{\mathrm{PPRIM}}(1^\lambda, u, x : K)}$
$y \leftarrow \mathsf{F_{Chop}.Ev}^{\mathsf{P}}(1^\lambda, K_u, x)$
Return y

$\underline{\mathsf{S.Exp}^{\mathrm{PPRIM}}(1^\lambda, u, T_u : K)}$
If $K_u = \bot$:
  If $T_u[\varepsilon] = \bot$: $T_u[\varepsilon] \leftarrow\!\!\$\ \mathsf{Chop.Out}(\lambda)$
  $K_u \leftarrow\!\!\$\ \mathsf{Chop}^{-1}(1^\lambda, T_u[\varepsilon])$
  For $x \neq \varepsilon$ s.t. $T_u[x] \neq \bot$:
    PPRIM$(\mathsf{Prog}, K_u, x, T_u[x])$
Return K_u

Our proof closely follows the steps of the proof of PRF security with leakage [5, Theorem 7.2, ePrint version] and SIM*-AC-PRF security [22, Theorem D.1] of random oracles.

We construct a series of hybrids G_0 to G_3 switching from the real world of SIM*-AC-PRF to the ideal world. Game G_0 modifies the real world to sample keys so that no two users have the same key and no m users have keys with the same leakage. A birthday bound and $\mathsf{P}^{\mathsf{coll}}_{\mathsf{Chop}}$ bound how detectable this change is. Additionally, PPRIM and EV are rewritten to separately sample their outputs at random with appropriate checks to make sure they are consistent. Game G_1 removes these consistency checks. This modification is only detectable if the adversary queries an unexposed user's key to PPRIM, which we can bound with $(m(\lambda)-1)p(\lambda)/\rho_{\mathsf{Chop}}(\lambda)$. Game G_2 returns to sampling all of the keys uniformly at random. Finally, G_3 switches from sampling uniformly random keys to first sampling the "leakage" of the key, then sampling the key using Chop^{-1}. This makes it equivalent to the ideal world. This switch is bounded by $u(\lambda) \cdot \varepsilon_{\mathsf{Chop}}(\lambda)$. The following claims will hold:

1. $\Pr[\mathsf{G}^{\mathsf{sim^*\text{-}ac\text{-}prf}}_{\mathsf{F_{Chop}},\mathsf{S},\mathsf{P_{rom}},\mathcal{A}_{\mathsf{prf}}}(\lambda) | b = 1] - \Pr[\mathsf{G}_0(\lambda)] \leq u(\lambda)^2/2|\mathsf{F_{Chop}.K}(\lambda)| + \mathsf{P}^{\mathsf{coll}}_{\mathsf{Chop}}(\lambda, u(\lambda), m(\lambda))$
2. $\Pr[\mathsf{G}_0(\lambda)] - \Pr[\mathsf{G}_1(\lambda)] \leq \frac{(m(\lambda)-1)p(\lambda)}{\rho_{\mathsf{Chop}}(\lambda)}$
3. $\Pr[\mathsf{G}_1(\lambda)] - \Pr[\mathsf{G}_2(\lambda)] \leq 0$
4. $\Pr[\mathsf{G}_2(\lambda)] - \Pr[\mathsf{G}_3(\lambda)] \leq u(\lambda) \cdot \varepsilon_{\mathsf{Chop}}(\lambda)$
5. $\Pr[\mathsf{G}_3(\lambda)] = \Pr[\neg \mathsf{G}^{\mathsf{sim^*\text{-}ac\text{-}prf}}_{\mathsf{F_{Chop}},\mathsf{S},\mathsf{P_{rom}},\mathcal{A}_{\mathsf{prf}}}(\lambda) | b = 0]$

Combining these (in)equalities gives the claimed bound.

Claim 1. Game G_0 is defined in Fig. 36. It contains highlighted and boxed code. For notational simplicity, we assume the name of the users queried by $\mathcal{A}_{\mathsf{prf}}$ are $1, \ldots, u(\lambda)$. Then we define $\mathcal{K}$ to be the set of key tuples $(K_1, \ldots, K_{u(\lambda)})$ that contain no key collisions between users and not $m(\lambda)$-wise collisions between the

Games $\boxed{\mathsf{G}_0(\lambda)}$,$\boxed{\mathsf{G}_1(\lambda)}$,$\mathsf{G}_2(\lambda)$

$\sigma_{\mathsf{rom}} \leftarrow [\cdot]$
$\mathcal{K} \leftarrow \{(K_1, \ldots, K_{u(\lambda)}) \ : \ \forall i \neq j, K_i \neq K_j \wedge$
$\quad \mu(\mathsf{Chop}(1^\lambda, K_1), \ldots, \mathsf{Chop}(1^\lambda, K_{u(\lambda)})) < m(\lambda)\}$
For $i = 1, \ldots, u(\lambda)$: $K_i \leftarrow\!\!\$\ \mathsf{F}_{\mathsf{Chop}}.\mathsf{K}(\lambda)$
If $(K_1, \ldots, K_{u(\lambda)}) \notin \mathcal{K}$:
  $\mathsf{bad}_{\mathsf{coll}} \leftarrow \mathsf{true}$; $\boxed{(K_1, \ldots, K_{u(\lambda)}) \leftarrow\!\!\$\ \mathcal{K}}$
$b' \leftarrow\!\!\$\ \mathcal{A}_{\mathsf{prf}}^{\mathrm{Ev},\mathrm{Exp},\mathrm{PP}_{\mathrm{RIM}}}(1^\lambda)$
Return $(b' = 1)\boxed{\wedge \neg \mathsf{bad}_{\mathsf{coll}}}$

PPRIM$(\mathsf{Op}, (k, x), y)$
If $\mathsf{Op} = \mathsf{Ls}$: $y \leftarrow\!\!\$\ \mathsf{F}.\mathsf{Out}(\lambda)$
If $\exists u$ s.t. $\neg\mathsf{corr}_u$ and $K_u = k$ and $T_u[x] \neq \perp$:
  $\mathsf{bad}_{\mathsf{guess}} \leftarrow \mathsf{true}$; $\sigma_{\mathsf{rom}}[(k, x)] \leftarrow T_u[x]$
If $\sigma_{\mathsf{rom}}[(k, x)] = \perp$: $\sigma_{\mathsf{rom}}[(k, x)] \leftarrow y$
If $\mathsf{Op} = \mathsf{Ls}$: Return $\sigma_{\mathsf{rom}}[(k, x)]$
Return $\perp$

Ev(u, x)
If $x = \varepsilon$: $T_u[x] \leftarrow \mathsf{Chop}(1^\lambda, K_u)$
Else if corr_u: $T_u[x] \leftarrow\!\!\$\ \mathsf{P}_{\mathsf{rom}}.\mathsf{Ls}(1^\lambda, (K_u, x) : \sigma_{\mathsf{rom}})$
Else if $T_u[x] \neq \perp$: $T_u[x] \leftarrow T_u[x]$ // No-op
Else if $\sigma_{\mathsf{rom}}[(K_u, x)] \neq \perp$:
  $\mathsf{bad}_{\mathsf{guess}} \leftarrow \mathsf{true}$; $T_u[x] \leftarrow \sigma_{\mathsf{rom}}[(K_u, x)]$
Else $T_u[x] \leftarrow\!\!\$\ \mathsf{F}.\mathsf{Out}(\lambda)$
Return $T_u[x]$

Exp(u)
If $\neg\mathsf{corr}_u$:
  For $x \neq \varepsilon$ s.t. $T_u[x] \neq \perp$:
    $\mathsf{P}_{\mathsf{rom}}.\mathsf{Prog}(1^\lambda, (K_u, x), T_u[x] : \sigma_{\mathsf{rom}})$
$\mathsf{corr}_u \leftarrow \mathsf{true}$
Return K_u

Fig. 36. Hybrid games G_0, G_1, and G_2 for proving security of $\mathsf{F}_{\mathsf{Chop}}$

Games $\mathsf{G}_{2.5}(\lambda)$,$\mathsf{G}_3(\lambda)$

$\sigma_{\mathsf{rom}} \leftarrow\!\!\$\ \mathsf{P}_{\mathsf{rom}}.\mathsf{Init}(1^\lambda)$
For $i = 1, \ldots, u(\lambda)$:
  $K_i \leftarrow\!\!\$\ \mathsf{F}_{\mathsf{Chop}}.\mathsf{K}(\lambda)$
  $\ell_i \leftarrow\!\!\$\ \mathsf{Chop}.\mathsf{Out}(\lambda)$
  $K_i \leftarrow\!\!\$\ \mathsf{Chop}^{-1}(1^\lambda, \ell_i)$
$b' \leftarrow\!\!\$\ \mathcal{A}_{\mathsf{prf}}^{\mathrm{Ev},\mathrm{Exp},\mathrm{PP}_{\mathrm{RIM}}}(1^\lambda)$
Return $(b' = 1)$

PPRIM$(\mathsf{Op}, (k, x), y)$
$y \leftarrow\!\!\$\ \mathsf{P}_{\mathsf{rom}}.\mathsf{Op}(1^\lambda, (k, x), y : \sigma_{\mathsf{rom}})$
Return y

Ev(u, x)
If $x = \varepsilon$: $T_u[x] \leftarrow \mathsf{Chop}(1^\lambda, K_u)$
Else if corr_u: $T_u[x] \leftarrow\!\!\$\ \mathsf{P}_{\mathsf{rom}}.\mathsf{Ls}(1^\lambda, (K_u, x) : \sigma_{\mathsf{rom}})$
Else if $T_u[x] \neq \perp$: $T_u[x] \leftarrow T_u[x]$ // No-op
Else $T_u[x] \leftarrow\!\!\$\ \mathsf{F}.\mathsf{Out}(\lambda)$
Return $T_u[x]$

Exp(u)
If $\neg\mathsf{corr}_u$:
  For $x \neq \varepsilon$ s.t. $T_u[x] \neq \perp$:
    $\mathsf{P}_{\mathsf{rom}}.\mathsf{Prog}(1^\lambda, (K_u, x), T_u[x] : \sigma_{\mathsf{rom}})$
$\mathsf{corr}_u \leftarrow \mathsf{true}$
Return K_u

Fig. 37. Hybrid games $\mathsf{G}_{2.5}$ and G_3 for proving security of $\mathsf{F}_{\mathsf{Chop}}$

"leakages" of the keys. If the randomly sampled keys are not in $\mathcal{K}$ we re-sample them from $\mathcal{K}$ and set a bad flag with ensures that the game returns false.

Additionally, we've written out the code of PPRIM to explicitly sample the random function using σ_{rom} and Ev to sample its random outputs using T_u for unexposed users. We need these random tables to define a single consistent random function, so we maintain the invariant that $T_u[x] = \sigma_{\mathsf{rom}}[(K_u, x)]$ whenever both these values are non-$\perp$ for unexposed users. Because users are guaranteed to have distinct keys, we do not need to account for consistency between different users' T_u tables. When users are exposed, we program $\sigma_{\mathsf{rom}}[(K_u, x)]$ to equal $T_u[x]$ wherever the latter is non-$\perp$ and thereafter use $\mathsf{P}_{\mathsf{rom}}$ with σ_{rom} directly in Ev. These invariants ensure the behavior of these oracles matches the real world.

So we get $\Pr[\mathsf{G}^{\mathsf{sim}^*\text{-}\mathsf{ac}\text{-}\mathsf{prf}}_{\mathsf{F}_{\mathsf{Chop}},\mathsf{S},\mathsf{P}_{\mathsf{rom}},\mathcal{A}_{\mathsf{prf}}}(\lambda) | b = 1] - \Pr[\mathsf{G}_0(\lambda)] \leq \Pr[\mathsf{G}_0(\lambda) \text{ sets } \mathsf{bad}_{\mathsf{coll}}]$. Then the claim follows from $\Pr[\mathsf{bad}_{\mathsf{coll}}] \leq \binom{u(\lambda)}{2} / |\mathsf{F}_{\mathsf{Chop}}.\mathsf{K}(\lambda)| + \mathsf{P}^{\mathsf{coll}}_{\mathsf{Chop}}(\lambda, u(\lambda), m(\lambda))$ where these terms come, respectively, from the probability of two user's sampling the same key and the probability of $m(\lambda)$ users having the same key leakage.

Claim 2. Game G_1 is also defined in Fig. 36. It contains boxed, but not highlighted code. It differs from G_0 in removing code which occurs after $\mathsf{bad}_{\mathsf{guess}}$ is set so $\Pr[\mathsf{G}_0(\lambda)] - \Pr[\mathsf{G}_1(\lambda)] \leq \Pr[\mathsf{G}_1 \text{ sets } \mathsf{bad}_{\mathsf{guess}}]$. This bad flag corresponds to the checks that were keeping σ_{rom} and T_u consistent for unexposed users; it can only be

set if PPRIM is queried with a k equal to the key of an exposed user. (Note that $\mathsf{P}_{\mathsf{rom}}.\mathsf{Prog}$ and $\mathsf{P}_{\mathsf{rom}}.\mathsf{Ls}$ in EXP and EV can add values to σ_{rom}, but only using exposed users' keys which don't equal other users' keys.)

Suppose the adversary queries PPRIM with a k such that $\mathsf{Chop}(1^\lambda, k) = \ell$. Define sets $S \supseteq S' \supseteq S''$ by $S = \{\, k : \mathsf{Chop}(1^\lambda, k) = \ell \,\}$, $S' = \{\, K_u : \mathsf{Chop}(1^\lambda, K_u) = \ell \,\}$ and $S'' = \{\, K_u : \mathsf{Chop}(1^\lambda, K_u) = \ell \wedge \mathsf{corr}_u \,\}$. We want to bound the probability that $k \in S' \setminus S''$. From the perspective of the adversary, each key in $S' \setminus S''$ is uniformly randomly distributed in $S \setminus S''$ so we can bound this by $(|S'| - |S''|)/(|S| - |S''|) \leq |S'|/|S| \leq (m(\lambda) - 1)/\rho_{\mathsf{Chop}}(\lambda)$. The first inequality follows because $(x - n)/(y - n) \leq x/y$ whenever $y \geq x$ and $y > n \geq 0$. The second inequality follows because the sampling of the keys ensures $|S'| \leq m(\lambda) - 1$ and $|S| \geq \rho_{\mathsf{Chop}}(\lambda)$ by definition. A union bound over the $p(\lambda)$ queries to PPRIM gives the claim.

Claim 3. Game G_2 is also defined in Fig. 36. It contains neither boxed nor highlighted code. Because G_1 always returns false when $\mathsf{bad}_{\mathsf{coll}}$ was set, the removal of boxed code can only increase the probability G_2 returns true, giving the claim.

Claim 4. Game $\mathsf{G}_{2.5}$ is defined in Fig. 37. It does not contain highlighted code. It was obtained from G_2 by removing unused if statements in PPRIM and EV and by simplifying PPRIM to just call $\mathsf{P}_{\mathsf{rom}}$. This is an equivalent rewriting of G_2. Hence $\Pr[\mathsf{G}_2(\lambda)] = \Pr[\mathsf{G}_{2.5}(\lambda)]$.

Now G_3 in the same figure contains the highlighted code. If differs from $\mathsf{G}_{2.5}$ only in that rather than sampling keys uniformly, we instead sample their "leakage" ℓ_i at random and then use Chop^{-1}. Applying a $\varepsilon_{\mathsf{Chop}}(\lambda)$ bound for each key gives the claim.

Claim 5. Finally we claim G_3 is identical to the ideal world with simulator S, except with flipped output. Essentially, the ideal world just delays sampling ℓ_i until it is needed in EV or EXP and delays sampling K_i from ℓ_i until it is needed in EXP. Correctness of EV uses in part that $\mathsf{Chop}(1^\lambda, \mathsf{Chop}^{-1}(1^\lambda, \ell)) = \ell$. □

G.2 Security of Davies-Meyer Compression Function with Leakage

In this section, we will look at when the compression function is built from a blockcipher using the Davies-Mayer construction. We analyze this in the ideal cipher model as captured by $\mathsf{P}_{\mathsf{icm}}$ defined below, Here we use the convention that setting $\sigma_k^+[x] \leftarrow z$ also sets $\sigma_k^-[z] \leftarrow x$ and vice versa. We define $\overline{\circ} = -$ if $\circ = +$ and vice versa.

$$\begin{array}{l|l|l}
\underline{\mathsf{P}_{\mathsf{icm}}.\mathsf{Init}(1^\lambda)} & \underline{\mathsf{P}_{\mathsf{icm}}.\mathsf{Ls}(1^\lambda, (k, x, \circ) : \sigma)} \;\; /\!/ \circ \in \{+, -\} & \underline{\mathsf{P}_{\mathsf{icm}}.\mathsf{Prog}(1^\lambda, (k, x, \circ), z)} \;\; /\!/ \circ \in \{+, -\} \\
\text{Return } \sigma^+ = [\cdot] & \text{If } \sigma_k^\circ[x] = \bot\text{:} & \text{If } \sigma_k^\circ[x] = \bot \text{ and } \sigma^{\overline{\circ}}[k, z] = \bot\text{:} \\
 & \quad \sigma_k^\circ[x] \leftarrow\!\!\$\; \mathcal{R}_\lambda \setminus \{\sigma_k^\circ[x'] : \sigma_k^\circ[x'] \neq \bot\} & \quad \sigma_k^\circ[x] \leftarrow z \\
 & \text{Return } \sigma_k^\circ[x] & \text{Return } \bot
\end{array}$$

Here $(k, x, +)$ indicates a forward invocation of the ideal cipher with key k and input x while $(k, x, -)$ indicates a backwards query.

The Davies-Meyer construction DM is defined by $\mathsf{DM}.\mathsf{Ev}^{\mathsf{P}_{\mathsf{icm}}}(1^\lambda, K, x) = \mathsf{P}_{\mathsf{icm}}(x, K, +) \oplus K$ with $\mathsf{DM}.\mathsf{Inp} = \mathsf{DM}.\mathsf{Out} = \mathcal{R}$. Note the key of DM is used as an input of $\mathsf{P}_{\mathsf{icm}}$ and the input of DM is used as a key of $\mathsf{P}_{\mathsf{icm}}$.

Theorem 9. *Let* $\mathsf{Chop}(1^\lambda, \cdot) : \mathsf{DM}.\mathsf{K}(\lambda) \to \mathsf{Chop}.\mathsf{Out}(\lambda)$ *and* $m(\lambda) \geq 1$. *Then we can build an efficient* S *such that*

$$\mathsf{Adv}^{\mathsf{sim}^*\text{-}\mathsf{ac}\text{-}\mathsf{prf}}_{\mathsf{DM}_{\mathsf{Chop}}, \mathsf{S}, \mathsf{P}, \mathcal{A}_{\mathsf{prf}}}(\lambda) \leq \frac{u(\lambda)(2q(\lambda) + u(\lambda))}{2|\mathsf{DM}_{\mathsf{Chop}}.\mathsf{K}(\lambda)|} + \mathsf{P}^{\mathsf{coll}}_{\mathsf{Chop}}(\lambda, u(\lambda), m(\lambda)) + \frac{(u(\lambda) + m(\lambda) - 1)p(\lambda)}{\rho_{\mathsf{Chop}}(\lambda)} + u(\lambda) \cdot \varepsilon_{\mathsf{Chop}}(\lambda)$$

holds for all $\mathcal{A}_{\mathsf{prf}}$ *making* $p(\lambda)$ *queries to* PPRIM, $q(\lambda)$ *queries to* EV, *and querying at most* $u(\lambda)$ *distinct users.*

The dependence on up/ρ_{Chop} do not appear the bound claimed for IND-PRF security with leakage in [5]. They both arise in our proof from patching errors we identified in the original proof. This term is a meaningful degradation in security that obviates the "better than trivial" multi-user security desired of the proof. We discuss the source of the errors and whether this term might be improved after our proof.

Proof. Our proof is somewhat analogous to the proof of Theorem 8, but the use of an ideal cipher rather than a random oracle adds additional complexity because we just account for collisions with both the inputs and the outputs of the primitive. We define our simulator S as follows.

$\underline{\mathsf{S.Init}(1^\lambda)}$
$K_{(\cdot)} \leftarrow [\cdot]$
Return $K_{(\cdot)}$

$\underline{\mathsf{S.Ev}^{\text{PPRIM}}(1^\lambda, u, x : K)}$
$y \leftarrow \mathsf{DM}_{\mathsf{Chop}}.\mathsf{Ev}^{\mathsf{P}}(1^\lambda, K_u, x)$
Return y

$\underline{\mathsf{S.Exp}^{\text{PPRIM}}(1^\lambda, u, T_u : K)}$
If $K_u = \bot$:
  If $T_u[\varepsilon] = \bot$: $T_u[\varepsilon] \leftarrow\!\!\$\ \mathsf{Chop.Out}(\lambda)$
  $K_u \leftarrow\!\!\$\ \mathsf{Chop}^{-1}(1^\lambda, T_u[\varepsilon])$
  For $x \neq \varepsilon$ s.t. $T_u[x] \neq \bot$:
    $\text{PPRIM}(\mathsf{Prog}, (x, K_u, +), K_u \oplus T_u[x])$
Return K_u

We construct a series of hybrids G_0 to G_3 switching from the real world of SIM*-AC-PRF to the ideal world. Game G_0 modifies the real world to sample keys so that no two users have the same key and no m users have keys with the same leakage. A birthday bound and $\mathsf{P}^{\mathsf{coll}}_{\mathsf{Chop}}$ bound how detectable this change is. Additionally, PPRIM and EV are rewritten to separately sample their outputs at random with appropriate checks to make sure they are consistent. Game G_1 removes these consistency checks. Roughly, this modification is only detectable if the adversary queries an unexposed user's key K_u to PPRIM, which we can bound with $(m(\lambda)-1)p(\lambda)/\rho_{\mathsf{Chop}}(\lambda)$ or it queries an output value z satisfying $z \oplus \text{EV}(u,x) = K_u$ which we can bound with $u(\lambda)p(\lambda)/\rho_{\mathsf{Chop}}(\lambda)$, or EV samples a random output which has already been used as an output of the ideal cipher which we can bound with $q(\lambda)\cdot(q(\lambda)+p(\lambda))/|\mathcal{R}_\lambda|$ Game G_2 returns to sampling all of the keys uniformly at random. Finally, G_3 switches from sampling uniformly random keys to first sampling the "leakage" of the key, then sampling the key using Chop^{-1}. This makes it equivalent to the ideal world. This switch is bounded by $u(\lambda)\cdot\varepsilon_{\mathsf{Chop}}(\lambda)$. The following claims will hold:

1. $\Pr[\mathsf{G}^{\mathsf{sim^*\text{-}ac\text{-}prf}}_{\mathsf{DM}_{\mathsf{Chop}},\mathsf{S},\mathsf{P},\mathcal{A}_{\mathsf{prf}}}(\lambda)|b=1] - \Pr[\mathsf{G}_0(\lambda)] \leq u(\lambda)^2/2|\mathsf{DM}_{\mathsf{Chop}}.\mathsf{K}(\lambda)| + \mathsf{P}^{\mathsf{coll}}_{\mathsf{Chop}}(\lambda, u(\lambda), m(\lambda))$
2. $\Pr[\mathsf{G}_0(\lambda)] - \Pr[\mathsf{G}_1(\lambda)] \leq (m(\lambda)-1)p(\lambda)/\rho_{\mathsf{Chop}}(\lambda) + u(\lambda)p(\lambda)/\rho_{\mathsf{Chop}}(\lambda) + u(\lambda)q(\lambda)/|\mathsf{DM}_{\mathsf{Chop}}.\mathsf{K}(\lambda)|$
3. $\Pr[\mathsf{G}_1(\lambda)] - \Pr[\mathsf{G}_2(\lambda)] \leq 0$
4. $\Pr[\mathsf{G}_2(\lambda)] - \Pr[\mathsf{G}_3(\lambda)] \leq u(\lambda)\cdot\varepsilon_{\mathsf{Chop}}(\lambda)$
5. $\Pr[\mathsf{G}_3(\lambda)] = \Pr[\neg\mathsf{G}^{\mathsf{sim^*\text{-}ac\text{-}prf}}_{\mathsf{DM}_{\mathsf{Chop}},\mathsf{S},\mathsf{P},\mathcal{A}_{\mathsf{prf}}}(\lambda)|b=0]$

Combining these (in)equalities gives the claimed bound.

Claim 1. Game G_0 is defined in Fig. 38. It contains highlighted and boxed code. For notational simplicity, we assume the name of the users queried by $\mathcal{A}_{\mathsf{prf}}$ are $1,\ldots,u(\lambda)$. Then we define $\mathcal{K}$ to be the set of key tuples $(K_1,\ldots,K_{u(\lambda)})$ that contain no key collisions between users and not $m(\lambda)$-wise collisions between the "leakages" of the keys. If the randomly sampled keys are not in $\mathcal{K}$ we re-sample them from $\mathcal{K}$ and set a bad flag with ensures that the game returns false.

Additionally, we have divided the lazy sampling of the ideal cipher between state σ used in PPRIM and table V used in EV. Not that when a user is exposed, the values from $V_x[u]$ are copied into σ and σ is henceforth always used for it. Together they (partially) define a keyed-permutation Π by

$$\Pi_x(k) = \begin{cases} \sigma^+_x[k], & \text{if } \sigma^+_x[k] \neq \bot \\ V_x[u], & \text{if } V_x[u] \neq \bot \text{ and } K = K_u \\ \bot, & \text{otherwise.} \end{cases}$$

For this to be well-defined the checks need to ensure that the first two cases above always give consistent answers. Inconsistent answers could arise from:

1. $\sigma^+_x[k] \neq V_x[u]$ where $K = K_u$ and $\bot \notin \{\sigma^+_x[k], V_x[u]\}$
2. $\sigma^+_x[k] = V_x[u]$ where $K \neq K_u$ and $\bot \notin \{\sigma^+_x[k], V_x[u]\}$
3. $V_x[u] \neq V_x[v]$ where $K_u = K_v$ and $\bot \notin \{V_x[u], V_x[v]\}$
4. $V_x[u] = V_x[v]$ where $K_u \neq K_v$ and $\bot \notin \{V_x[u], V_x[v]\}$.

Games $\boxed{\mathsf{G}_0(\lambda)}$,$\boxed{\mathsf{G}_1(\lambda)}$,$\mathsf{G}_2(\lambda)$

$\sigma \leftarrow [\cdot]$
$\mathcal{K} \leftarrow \{(K_1,\ldots,K_{u(\lambda)}) \ : \ \forall i \neq j, K_i \neq K_j \wedge$
$\quad \mu(\mathsf{Chop}(1^\lambda,K_1),\ldots,\mathsf{Chop}(1^\lambda,K_{u(\lambda)})) < m(\lambda)\}$
For $i = 1,\ldots,u(\lambda)$: $K_i \leftarrow\!\!\$\ \mathsf{DM}_{\mathsf{Chop}}.\mathsf{K}(\lambda)$
If $(K_1,\ldots,K_{u(\lambda)}) \notin \mathcal{K}$:
  $\mathsf{bad}_{\mathsf{coll}} \leftarrow \mathsf{true}$; $\boxed{(K_1,\ldots,K_{u(\lambda)}) \leftarrow\!\!\$\ \mathcal{K}}$
$b' \leftarrow\!\!\$\ \mathcal{A}_{\mathsf{prf}}^{\mathrm{Ev},\mathrm{Exp},\mathrm{PPRIM}}(1^\lambda)$
Return $(b'=1)\boxed{\wedge\neg\mathsf{bad}_{\mathsf{coll}}}$

PPRIM$(\mathsf{Ls},(x,k,+))$
If $\exists u$ s.t. $\neg\mathsf{corr}_u$ and $k = K_u$:
  $\mathsf{bad}_{\mathsf{guess}} \leftarrow \mathsf{true}$; If $V_x[u] \neq \bot$: $\sigma_x^+[k] \leftarrow V_x[u]$
If $\sigma_x^+[k] = \bot$: $\sigma_x^+[k] \leftarrow\!\!\$\ \mathcal{R}_\lambda \setminus \sigma.\mathsf{Range}(x)$
  If $\exists u$ s.t. $\neg\mathsf{corr}_u$ and $V_x[u] = \sigma_x^+[k]$:
    $\mathsf{bad}_{\mathsf{guess2}} \leftarrow \mathsf{true}$; $\sigma_x^+[k] \leftarrow\!\!\$\ \mathcal{R}_\lambda \setminus \mathsf{Range}(x)$
Return $\sigma_x^+[k]$

PPRIM$(\mathsf{Prog},(x,k,\circ),z)$
If $\circ = -$: $(k,z) \leftarrow (z,k)$
If $\exists u$ s.t. $\neg\mathsf{corr}_u$ and $k = K_u$:
  $\mathsf{bad}_{\mathsf{guess}} \leftarrow \mathsf{true}$; If $V_x[u] \neq \bot$: $\sigma_x^+[k] \leftarrow V_x[u]$
If $\exists u$ s.t. $\neg\mathsf{corr}_u$ and $V_x[u] = z$:
  $\mathsf{bad}_{\mathsf{guess2}} \leftarrow \mathsf{true}$; $\sigma_x^-[z] \leftarrow K_u$
If $\sigma_x^+[k] = \bot$ and $\sigma_x^-[z] = \bot$: $\sigma_x^+[k] \leftarrow z$
Return $\bot$

Ev(u,x)
If $x = \varepsilon$: Return $\mathsf{Chop}(1^\lambda,K_u)$
Else if $\mathsf{corr}_u = \mathsf{true}$: $V_x[u] \leftarrow\!\!\$\ \mathsf{P}_{\mathsf{icm}}.\mathsf{Ls}((x,K_u,+):\sigma)$
Else if $V_x[u] \neq \bot$: $V_x[u] \leftarrow V_x[u]$ // No-op
Else if $\sigma_x^+[K_u] \neq \bot$: $\mathsf{bad}_{\mathsf{guess}} \leftarrow \mathsf{true}$; $V_x[u] \leftarrow \sigma_x^+[K_u]$
Else $V_x[u] \leftarrow\!\!\$\ \mathcal{R}_\lambda$
  If $\exists v \neq u$ s.t. $V_x[v] = V_x[u]$:
    $\mathsf{bad}_\$ \leftarrow \mathsf{true}$; $V_x[u] \leftarrow\!\!\$\ \mathcal{R}_\lambda \setminus \mathsf{Range}(x)$
  Else if $\exists k$ s.t. $\sigma_x^+[k] = V_x[u]$:
    $\mathsf{bad}_{\mathsf{guess2}} \leftarrow \mathsf{true}$; $V_x[u] \leftarrow\!\!\$\ \mathcal{R}_\lambda \setminus \mathsf{Range}(x)$
$T_u[x] \leftarrow K_u \oplus V_x[u]$
Return $T_u[x]$

PPRIM$(\mathsf{Ls},(x,z,-))$
If $\exists u$ s.t. $\neg\mathsf{corr}_u$ and $V_x[u] = z$:
  $\mathsf{bad}_{\mathsf{guess2}} \leftarrow \mathsf{true}$; $\sigma_x^-[z] \leftarrow K_u$
If $\sigma_x^-[z] = \bot$: $\sigma_x^-[z] \leftarrow\!\!\$\ \mathcal{R}_\lambda \setminus \sigma.\mathsf{Domain}(x)$
  If $\exists u$ s.t. $\neg\mathsf{corr}_u$ and $\sigma_x^-[z] = K_u$:
    $\mathsf{bad}_{\mathsf{guess}} \leftarrow \mathsf{true}$
    If $V_x[u] \neq \bot$: $\sigma_x^-[z] \leftarrow\!\!\$\ \mathcal{R}_\lambda \setminus \mathsf{Domain}(x)$
Return $\sigma_x^-[z]$

Exp(u)
If $\neg\mathsf{corr}_u$: For all $x \neq \varepsilon$ s.t. $T_u[x] \neq \bot$:
  $\mathsf{P}_{\mathsf{icm}}.\mathsf{Prog}(1^\lambda,(x,K_u,+),V_x[u]:\sigma)$
$\mathsf{corr}_u \leftarrow \mathsf{true}$; Return K_u

Definitions // $x \neq \varepsilon$
$\mathsf{Range}(x) = \{\, \sigma_x^+[k] : \sigma_x^+[k] \neq \bot \,\} \cup \{\, V_x[u] : V_x[u] \neq \bot \,\}$
$\mathsf{Domain}(x) = \{\, k : \sigma_x^+[k] \neq \bot \,\} \cup \{\, K_u : V_x[u] \neq \bot \,\}$
$\sigma.\mathsf{Range}(x) = \{\, \sigma_x^+[k] : \sigma_x^+[k] \neq \bot \,\}$
$\sigma.\mathsf{Domain}(x) = \{\, k : \sigma_x^+[k] \neq \bot \,\}$

Fig. 38. Hybrid games G_0, G_1, and G_2 for proving security of $\mathsf{DM}_{\mathsf{Chop}}$

Inconsistency (3) can never arise because we are sampling all user keys to be distinct. The consistency checks associated with $\mathsf{bad}_{\mathsf{guess}}$ ensure that (1) will not occur. The consistency checks associated with $\mathsf{bad}_{\mathsf{guess2}}$ ensure that (2) will not occur. The consistency checks associated with $\mathsf{bad}_\$$ ensures that (4) will not occur.

So we get $\Pr[\mathsf{G}^{\mathsf{sim}^*\text{-}\mathsf{ac}\text{-}\mathsf{prf}}_{\mathsf{DM}_{\mathsf{Chop}},\mathsf{S},\mathsf{P},\mathcal{A}_{\mathsf{prf}}}(\lambda)|b=1] - \Pr[\mathsf{G}_0(\lambda)] \leq \Pr[\mathsf{G}_0(\lambda) \text{ sets } \mathsf{bad}_{\mathsf{coll}}] \leq \binom{u(\lambda)}{2}/|\mathsf{DM}_{\mathsf{Chop}}.\mathsf{K}(\lambda)| + \mathsf{P}^{\mathsf{coll}}_{\mathsf{Chop}}(\lambda,u(\lambda),m(\lambda))$ where these terms come, respectively, from the probability of two user's sampling the same key and the probability of $m(\lambda)$ users having the same key leakage.

Claim 2. Game G_1 is also defined in Fig. 38. It contains boxed, but not highlighted code. It differs from G_0 in removing code which occurs after $\mathsf{bad}_\$$, $\mathsf{bad}_{\mathsf{guess}}$, $\mathsf{bad}_{\mathsf{guess2}}$ is set. corresponding to σ and V_x no longer jointly defining a keyed-permutation. We have $\Pr[\mathsf{G}_0(\lambda)] - \Pr[\mathsf{G}_1(\lambda)] \leq \Pr[\mathsf{G}_1 \text{ sets } \mathsf{bad}_\$ \vee \mathsf{bad}_{\mathsf{guess}} \vee \mathsf{bad}_{\mathsf{guess2}}]$.

Table V_x stores at most $u(\lambda)$ non-$\bot$ values. A union bound over these and the number of Ev queries gives $\Pr[\mathsf{G}_1 \text{ sets } \mathsf{bad}_\$] \leq q(\lambda)u(\lambda)/|\mathcal{R}_\lambda| = q(\lambda)u(\lambda)/|\mathsf{DM}_{\mathsf{Chop}}.\mathsf{K}(\lambda)|$.

We will analyze the probability that an individual PPRIM query can set the bad flags and then perform a union bound over all queries. Note that in G_1, until a user has been corrupted the view of the adversary only depends on its key via $\mathsf{Chop}(1^\lambda,K_u)$ values returned by evaluation.[8]

Consider $\mathsf{bad}_{\mathsf{guess}}$ for a forward lazy sampling query PPRIM$(\mathsf{Ls},(x,k,+))$ or programming query. Analogously to the analysis of $\mathsf{bad}_{\mathsf{guess}}$ for Theorem 8, there are fewer than m keys K_u satisfying $\mathsf{Chop}(1^\lambda,K_u) =$

[8] While $T_u[x] \leftarrow V_x[u] \oplus K_u$ in Ev, this does not cause a dependence because $V_x[u]$ is uniformly random and not used elsewhere.

Games $\mathsf{G}_{2.5}(\lambda)$,$\mathsf{G}_3(\lambda)$

$\sigma \leftarrow\!\!\$\ \mathsf{P}_{\mathsf{icm}}.\mathsf{Init}(1^\lambda)$
For $i = 1, \ldots, u(\lambda)$:
  $K_i \leftarrow\!\!\$\ \mathsf{DM}_{\mathsf{Chop}}.\mathsf{K}(\lambda)$
  $\ell_i \leftarrow\!\!\$\ \mathsf{Chop}.\mathsf{Out}(\lambda)$
  $K_i \leftarrow\!\!\$\ \mathsf{Chop}^{-1}(1^\lambda, \ell_i)$
$b' \leftarrow\!\!\$\ \mathcal{A}_{\mathsf{prf}}^{\textsc{Ev},\textsc{Exp},\textsc{PPrim}}(1^\lambda)$
Return $(b' = 1)$

PPrim$(\mathsf{Op}, (k, x), y)$
$y \leftarrow\!\!\$\ \mathsf{P}_{\mathsf{icm}}.\mathsf{Op}(1^\lambda, (k, x), y : \sigma)$
Return y

Ev(u, x)
If $x = \varepsilon$: $T_u[x] \leftarrow \mathsf{Chop}(1^\lambda, K_u)$
Else if $\mathsf{corr}_u = \mathsf{true}$: $T_u[x] \leftarrow\!\!\$\ K_u \oplus \mathsf{P}_{\mathsf{icm}}.\mathsf{Ls}((x, K_u, +) : \sigma)$
Else if $T_u[x] \neq \bot$: $T_u[x] \leftarrow T_u[x]$ // No-op
Else $T_u[x] \leftarrow\!\!\$\ \mathcal{R}_\lambda$
Return $T_u[x]$

Exp(u)
If $\neg\mathsf{corr}_u$:
  For all $x \neq \varepsilon$ s.t. $T_u[x] \neq \bot$:
    $\mathsf{P}_{\mathsf{icm}}.\mathsf{Prog}(1^\lambda, (x, K_u, +), K_u \oplus T_u[x] : \sigma)$
$\mathsf{corr}_u \leftarrow \mathsf{true}$
Return K_u

Fig. 39. Hybrid games $\mathsf{G}_{2.5}$ and G_3 for proving security of $\mathsf{DM}_{\mathsf{Chop}}$

$\mathsf{Chop}(1^\lambda, K)$ so by a union bound the probability this query sets $\mathsf{bad}_{\mathsf{guess}}$ is at most $(m(\lambda)-1)/\rho_{\mathsf{Chop}}(\lambda)$. The same logic with $\sigma_x^-[z]$ in place of k applies for an inverse query PPrim$(\mathsf{Ls}, (x, z, -))$. Note that $\mathsf{bad}_{\mathsf{guess}}$ can only be set by Ev if it was already set by a PPrim query, so we do not additionally need to analyze this.

Consider $\mathsf{bad}_{\mathsf{guess2}}$ for an inverse lazy sampling query PPrim$(\mathsf{Ls}, (x, z, -))$ or programming query. If $V_x[u] = z$, then $z \oplus T_u[x] = K_u$. By a union bound over all u, the probability this query sets $\mathsf{bad}_{\mathsf{guess2}}$ is at most $u(\lambda)/\rho_{\mathsf{Chop}}(\lambda)$. The same logic with $\sigma_x^+[k]$ in place of z applies for a forward query PPrim$(\mathsf{Ls}, (x, k, +))$. Technically, we only need to consider users previously queried via Ev(u, x) for the above. But then each later Ev(u, x) query has at most a $1/|\mathcal{R}_\lambda| \leq \rho_{\mathsf{Chop}}(\lambda)$ probability of $\mathsf{bad}_{\mathsf{guess2}}$ from this earlier primitive query, so we can fold them together to still get $u(\lambda)/\rho_{\mathsf{Chop}}(\lambda)$ overall.

Since the adversary makes at most p queries to PPrim, a union bound gives the claim that

$$\Pr[\mathsf{G}_0(\lambda)] - \Pr[\mathsf{G}_1(\lambda)] \leq \frac{(m(\lambda)-1)p(\lambda)}{\rho_{\mathsf{Chop}}(\lambda)} + \frac{u(\lambda)p(\lambda)}{\rho_{\mathsf{Chop}}(\lambda)} + \frac{u(\lambda)q(\lambda)}{|\mathsf{DM}_{\mathsf{Chop}}.\mathsf{K}(\lambda)|}.$$

Claim 3. Game G_2 is also defined in Fig. 38. It contains neither boxed nor highlighted code. Because G_1 always returns false when $\mathsf{bad}_{\mathsf{coll}}$ was set, the removal of boxed code can only increase the probability G_2 returns true, giving the claim.

Claim 4. Game $\mathsf{G}_{2.5}$ is defined in Fig. 39. It does not contain highlighted code. It was obtained from G_2 by removing unused if statements in PPrim and Ev and by simplifying PPrim to just call $\mathsf{P}_{\mathsf{icm}}$. This is an equivalent rewriting of G_2. Hence $\Pr[\mathsf{G}_2(\lambda)] = \Pr[\mathsf{G}_{2.5}(\lambda)]$.

Now G_3 in the same figure contains the highlighted code. If differs from $\mathsf{G}_{2.5}$ only in that rather than sampling keys uniformly, we instead sample their "leakage" ℓ_i at random and then use Chop^{-1}. Applying a $\varepsilon_{\mathsf{Chop}}(\lambda)$ bound for each key gives the claim.

Claim 5. Finally we claim G_3 is identical to the ideal world with simulator S, except with flipped output. Essentially, the ideal world just delays sampling ℓ_i until it is needed in Ev or Exp and delays sampling K_i from ℓ_i until it is needed in Exp. Correctness of Ev uses in part that $\mathsf{Chop}(1^\lambda, \mathsf{Chop}^{-1}(1^\lambda, \ell)) = \ell$. □

G.3 Proof Errors and Improving Concrete Security for Truncation

Bellare, et al. [5, Theorem 10.1, ePrint version] prove IND-PRF security of DM under key-leakage in the (non-programmable) ideal cipher model. The given proof is incorrect. It falsely claims that its game G_1 is equivalent to the real world of the IND-PRF game.

This game does not properly capture all of the four possible inconsistencies we discuss in Claim 1 of our proof. The consistency checks in G_1's PRF oracle cover inconsistencies (1), (3), and (4) in that order but

not (2). The consistency checks in G_1's primitive oracle for forward queries covers inconsistency (1) but not (2). The consistency checks in G_1's primitive oracle for backwards queries covers inconsistency (2) but not (1), though its code for avoiding (2) is incorrect as is the oracles return statement. Inconsistencies (3) and (4) do not apply to primitive queries.

The mp/ρ_{Chop} term in the proofs bound is claimed to cover (1) from the PRF and forward primitive queries and well as (2) from the backwards primitive queries. The last of these is seemingly incorrect, but the term should instead cover the missing (1) from these queries.

The up/ρ_{Chop} term in our bound arises from addressing inconsistency (2). Ideally we would like it to be an mp term, but the type of analysis used for (1) does not apply. We can think of the analysis in (1) as using the fact that for a given k and unexposed users' key K_u we have

$$\Pr[k = K_u] \leq \begin{cases} 1/\rho_{\mathsf{Chop}}, & \text{if } \mathsf{Chop}(1^\lambda, K_u) = \mathsf{Chop}(1^\lambda, k) \\ 0, & \text{otherwise.} \end{cases}$$

There are fewer than m keys satisfying $\mathsf{Chop}(1^\lambda, K_u) = \mathsf{Chop}(1^\lambda, k)$ and so a union bound gives $\Pr[k \in \{K_1, \ldots, K_{u(\lambda)}\}] \leq (m(\lambda) - 1)/\rho_{\mathsf{Chop}}(\lambda)$.

On the other hand, to analyze (2) for a given z and unexposed user u we have

$$\Pr[z = V_x[u] = T_u[x] \oplus K_u] \leq \begin{cases} 1/\rho_{\mathsf{Chop}}, & \text{if } \mathsf{Chop}(1^\lambda, K_u) = \mathsf{Chop}(1^\lambda, T_u[x] \oplus z) \\ 0, & \text{otherwise.} \end{cases}$$

In our proof we upper bound how many users can satisfy the first case of this by $u(\lambda)$. This gives $\Pr[z \in \{T_1[x] \oplus K_1, \ldots\}] \leq u(\lambda)/\rho_{\mathsf{Chop}}(\lambda)$. We do not know a good general argument for improving this, though can do so when $\mathsf{Chop}(1^\lambda, \cdot)$ is truncation.

The core observation is that truncation is a homomorphism over $\oplus$, so the first case implies that $\mathsf{Chop}(1^\lambda, K_u \oplus T_u[x]) = \mathsf{Chop}(1^\lambda, z)$. Consider a fixed x. Define $K_{u,x} = T_u[x] \oplus K_u$ for each user. Then $\mu(\mathsf{Chop}(1^\lambda, K_{1,x}), \ldots, \mathsf{Chop}(1^\lambda, K_{u(\lambda),x}))$ upper bounds the number of users that can satisfy the first case for any z. Because each $T_u[x]$ is uniform and independent, each $K_{1,x}$ is uniform and independent. Thus the probability there are at least m unexposed users satisfying the first case can be bounded by $\mathsf{P}^{\mathsf{coll}}_{\mathsf{Chop}}(u, m)$. Applying a union bound over the at most q values of x queried to Ev, the probability of their being an x with at least m unexposed users satisfying the first case is bounded by $q \cdot \mathsf{P}^{\mathsf{coll}}_{\mathsf{Chop}}(\lambda, u, m)$. Consequently, in Theorem 9 applied to truncation we can replace the up/ρ_{Chop} term in our bound with $q(\lambda) \cdot \mathsf{P}^{\mathsf{coll}}_{\mathsf{Chop}}(u, m) + (m-1)p/\rho_{\mathsf{Chop}}$, a significant improvement. For example, with the parameters Bellare, et al. consider (truncating from c to r bits and setting $m = 2u/2^r + 3cr$) we have $\mathsf{P}^{\mathsf{coll}}_{\mathsf{Chop}}(u, m) = e^{-c} < 2^{-c} = |\mathsf{DM}_{\mathsf{Chop}}.\mathsf{K}|$ so these two terms are upper-bounded by other terms in our theorem.