Two-Factor Authentication Can Harden Servers Against Offline Password Search

Xavier Boyen[1], Stanislaw Jarecki[2], Phillip Nazarian[2], Jiayu Xu[3], and Tianyu Zheng[4]

[1] Queensland University of Technology, Australia xavier.boyen@qut.edu.au
[2] University of California, Irvine, USA {sjarecki,pnazaria}@uci.edu
[3] Oregon State University, USA xujiay@oregonstate.edu
[4] University of California, Los Angeles, USA tyzheng@ucla.edu

Abstract. We propose a (universally composable) Two-Factor Authenticated Key Exchange (TFA-KE), a notion which extends asymmetric PAKE (aPAKE) with a 2nd authentication factor, a t-bit one-time code computed by a personal device based on a clock or a counter. Our notion strengthens the security of standard integration of an aPAKE with short authentication codes by slowing down offline brute-force password search in case of server compromise by a factor of 2^t. In other words, our TFA-KE notion uses t-bit authentication codes not only to improve on-line security of password authentication, as is the current practice, but also to strengthen password security on server corruption.

We show a generic framework for implementing TFA-KE, with two efficient instantiations. Our key enabling tool is a *tight one-way function* (TOWF) with an algebraic structure that allows for efficient function evaluation on secret-shared inputs. We initiate the study of such functions, and we provide two proposals which we show to be tightly one-way in the Generic Group Model. Tightness means that a function evaluation on an input sampled from domain $\mathcal{X}$ takes $\Omega(|\mathcal{X}|)$ time to invert, which in our application implies that offline password search attacks are slowed to $\Omega(|D| \cdot 2^t)$ for passwords sampled from dictionary D.

1 Introduction

Two-Factor Authentication. Passwords are the most common method of user verification across various digital systems. However, password re-use (see e.g. [40]), which means that a password compromise at one service implies the compromise of the same user at other services, together with effective low entropy of passwords chosen by a large fraction of users (see e.g. [7]), motivates adoption of 'two-factor' authentication, where in addition to a password the user proves that it has access to a personal device which can compute a one-time *authentication code* expected by the authentication server. Since the user typically reads the code displayed by her device, e.g. an authenticator app on her cell

* This is an extended version of the paper which appeared at EUROCRYPT 2026 [8].

phone, and must type it as a protocol input together with her password, authentication codes are short bitstrings of fixed length t, e.g. encoded as 6-digit numerical strings ($t = 20$) or 6-digit alphanumerical strings ($t = 30$).

Current two-factor authentication implementations decouple these two authentication components, i.e. password verification and authenticator code verification. Password verification can be implemented with any asymmetric Password Authenticated Key Exchange (aPAKE) [25,32], where the server holds a (salted) password hash rather than the password itself, while the authenticator code verification is typically implemented using a Pseudorandom Function R whose range is t-bit strings: The one-time authenticator code corresponding to time period or counter T is defined as $\mathsf{code}_T = R_{\mathsf{key}}(T)$ where key is a symmetric key shared by the user's personal device and the server, and the user and the server can follow any equality-testing protocol, ideally a (symmmetric) PAKE, to verify that they hold matching code values.[5]

The main benefit of two-factor authentication implemented as above is that it provides a second layer of defense to on-line authentication in the face of password weakness: Even if a user's password leaks, without access to the user's device the adversary can authenticate only with probability $1/2^t$, i.e. by guessing the correct authentication code. However, this second authentication factor does not help in protecting the password in the all-too-common case of server compromise: Since the server stores a password hash, the adversary can recover the password given the hash using a (brute-force) Offline Dictionary Attack (ODA), which takes time $\Omega(|D|)$ where D is the presumed dictionary from which the user picked her password. In other words, current integration of password and 2nd factor authentication creates a defense in case the user's password leaks but it doesn't help in protecting the password from leakage.

Two-Factor Authentication with 2^t-factor ODA slow-down. We show that two-factor authentication can improve *both online and offline* security of password authentication. Namely, we show Two-Factor Authenticated Key Exchange (TFA-KE) schemes where the server-held *password file* FILE allows for remote verification of the user's password pw and the correct t-bit authentication code code_T corresponding to time/counter T, but the brute-force search for pw which is possible if FILE is revealed due to a server compromise, requires $\Omega(|D| \cdot 2^t)$ computation instead of $\Omega(|D|)$ for password-only authentication.[6] Note that for the $t = 20$ or $t = 30$ cases mentioned above, our TFA-KE will slow

[5] For historical reasons, in today's practice on the internet, password authentication, with or without a second factor, merely consists of sending the user secrets across a PKI-authenticated SSL/TLS "secure channel", which places much of the security burden upon authenticating the server *prior* to password verification—completely missing out on the *raison d'être* of PAKE and especially aPAKE. To be clear, in this paper we do not assume any pre-existing secure channel nor PKI.

[6] Note that both bounds are optimal: In an aPAKE, password file FILE allows the adversary to offline test any password pw^* by locally running the aPAKE on client input pw^* and server input FILE, leading to ODA in time $\mathcal{O}(|D|)$. Likewise, in a TFA-KE, leaking FILE allows the adversary to offline test any $(\mathsf{pw}^*, \mathsf{code}^*)$ pair, by

ODA by considerable factors of respectively $2^{20} \approx 10^6$ or $2^{30} \approx 10^9$, and an increase of ODA time by a factor of 10^6 means that an attack which took a minute will now take 2 years.

What makes this problem non-trivial is that two-factor authentication must be secure under *lunch-time attack* on the user's personal device, i.e. a leaked code code_T corresponding to counter T cannot help to authenticate to a server who runs on any other counter $T^* \neq T$. This effectively implies that a code generation function $\mathsf{code}_T = R_{\mathsf{key}}(T)$ must be a PRF. (Value code_{T^*} must be unpredictable given any set of pairs (T, code_T) for $T \neq T^*$, and it must be indistinguishable from a random t-bit string to achieve a 2^t factor of security increase in on-line authentication.) Our question is therefore: Is there a PRF function R_{key} and a password file generation algorithm $\mathsf{Init}(\mathsf{pw}, \mathsf{key})$ which enable an efficient protocol Auth that authenticates client C and server S with respective inputs $(\mathsf{pw}, \mathsf{code})$ and $(\textsc{file}, T)$, if and only if $\textsc{file} = \mathsf{Init}(\mathsf{pw}, \mathsf{key})$ and $\mathsf{code} = R_{\mathsf{key}}(T)$? Moreover, leaking FILE must enable recovery of pw only via a brute-force search that runs Auth on all candidate pairs $(\mathsf{pw}^*, \mathsf{code}^*)$. If Init and R are arbitrary one-way functions then it is not even clear that you can solve it with (two-party) secure computation [45,27], because the combined inputs of both parties, $(\mathsf{pw}, \mathsf{code}, \textsc{file}, T)$, do not contain input key, which allows for efficient verification of this condition. On the other hand, if key is present in FILE then leakage of the latter would let an adversary compute $\mathsf{code}_T = R_{\mathsf{key}}(T)$ and run an ODA that tests password candidates using only $\mathcal{O}(|D|)$ computation.

Contribution I: Defining Two-Factor Authenticated Key Exchange. We propose a notion of Two-Factor Authenticated Key Exchange (TFA-KE), stated in the framework of universal composability (UC), which extends the UC asymmetric PAKE (aPAKE) model of Gentry, MacKenzie, and Ramzan [25] by the second authentication factor based on t-bit one-time authentication codes computed by the user's personal device. Our UC TFA-KE notion is a strict upgrade of UC aPAKE, and it defaults to UC aPAKE if $t = 0$. We model the authentication code generation function $\mathsf{code}_T = R(T)$ as a random function, which implies resilience to lunch-time attacks (in particular, an attacker who knows the correct password but does not know code_T can authenticate to either party with probability at most $1/2^t$), and we model an Offline Dictionary Attack (ODA) possible after compromise of the password file FILE as tests of explicit $(\mathsf{pw}^*, \mathsf{code}^*)$ candidates provided by the adversary, on some adversarially chosen counter T, which implies an $\Omega(|D| \cdot 2^t)$ computation bound on an ODA. We note that our model exposes the one-time code_T values to the environment, hence just like in the case of passwords, security in our TFA-KE model extends to environments that mistype them, mistakenly re-use them, etc.

Contribution II: Generic TFA-KE scheme which enables efficient instantiations. We show a generic framework for implementing TFA-KE, together with two efficient instantiations based on two proposals for *tight* one-way func-

running the TFA-KE on client input $(\mathsf{pw}^*, \mathsf{code}^*)$ and server input $(\textsc{file}, T)$. Since for any T there are 2^t possible code_T values, the ODA search takes $\mathcal{O}(|D| \cdot 2^t)$ time.

tions with algebraic properties (see Contribution III below). Our construction stems from the idea of Shirvanian et al. [42] which exhibited a TFA scheme with an $\Omega(|D| \cdot 2^t)$ bound on ODA after server compromise, but assuming a significantly simpler setting of Public Key Infrastructure (PKI). Password (and two-factor) authentication in the PKI setting reflects the current web authentication practice, where the client sends her password pw (and the one-time authenticator code) on an SSL channel authenticated by the server's PKI certificate.[7]

The idea of [42] was to set $\mathsf{key} = (k, s)$ where k is a key of an arbitrary PRF function, e.g. a block cipher, and s is a t-bit random pad, set $\mathsf{code}_T = R_{(k,s)}(T) = s \oplus \mathsf{PRF}_k(T)$, and let $\mathsf{Init}(\mathsf{pw}, (k, s))$ output $\text{FILE} = (k, \mathsf{H}(\mathsf{pw}, s))$ where H is a Random Oracle (RO) hash function. An authentication, which in the PKI setting is performed locally by the server on the local input FILE and message $(\mathsf{pw}, \mathsf{code})$ received on the SSL channel, parses $\text{FILE} = (k, F)$ and accepts iff $F = \mathsf{H}(\mathsf{pw}, \mathsf{code} \oplus \mathsf{PRF}_k(T))$. Note that $R_{(k,s)}$ is a PRF, and that the password file $(k, \mathsf{H}(\mathsf{pw}, s))$ allows for testing only explicitly provided (pw^*, s^*) candidates, implying an $\Omega(|D| \cdot 2^t)$ bound on ODA.

Our TFA-KE proposal starts from the above idea, and observes that pairs $(s_\mathsf{C}, s_\mathsf{S}) = (\mathsf{code}_T, \mathsf{PRF}_k(T))$ form a secret-sharing of pad s,[8] and that this sharing can be implemented with group addition if set $\{0,1\}^t$ forms an abelian group, e.g. $\mathbb{Z}_{2^t}$. We then generalize RO hash H to any *tight one-way* (and collision-resilient) function f (see below), and we use a *Conditional Key Encapsulation Mechanism* (CKEM) protocol to establish a secure key between the two parties if their respective inputs $(\mathsf{pw}, s_\mathsf{C})$ and (F, s_S) satisfy condition $F = f(\mathsf{pw}, s_\mathsf{C} + s_\mathsf{S})$. Further, to argue that $R_{(k,s)}$ acts like a random function even after server corruption we implement PRF_k using a hash function modeled as a programmable RO, and to enable extraction of an adversarial server's inputs we add a commitment to F, a check that CKEM runs on a committed value, and a non-interactive zero-knowledge proof of knowledge of f pre-image of the committed value.

The core of this TFA-KE protocol is the CKEM for a condition, denoted $\mathsf{SSEM}[f]$ for "secret-shared evaluation match", that $F = f(\mathsf{pw}, s)$ for a secret-shared s, client-held pw, and server-held committed F. This CKEM can be implemented using generic secure computation, but more efficient instantiations follow from tight one-way functions (OWF) f specifically designed to enable more efficient CKEM realizations, as we discuss below.

Contribution III: Tight OWF's with Algebraic Properties. We use the UC notion of a Tight One-Way Function (TOWF) of Bradley, Jarecki, and Xu [10], which models a one-way function f on domain $\mathcal{X}$ (which we re-formulate to include collision-resilience) s.t. for any subset S of $\mathcal{X}$, an adversary who receives $f(x)$ for x randomly sampled in S, must bear $\Omega(|S|)$ computation cost to recover x from $f(x)$. A Random Oracle hash is an example of a TOWF, but

[7] Relying on PKI in password and two-factor authentication has numerous security problems, including 'phishing attacks' where the client sends her password to an attacker who poses as the legitimate web service, and the fact that a spyware on the legitimate server can gain access to cleartext passwords sent to that server.

[8] It can be seen as a simple case of 2-out-of-2 pseudorandom sharing [16] of pad s.

instantiating f as a generic hash would allow only for a generic secure computation realization of CKEM for condition $\mathsf{SSEM}[f]$. Instead, we show two examples of TOWF's which allow for more efficient realizations of CKEM for $\mathsf{SSEM}[f]$.

If argument x were secret-shared over f domain[9] $\mathcal{X}$ then we could get an efficient CKEM for condition $\mathsf{SSEM}[f]$ if $f(x_{\mathsf{C}}) \cdot f(x_{\mathsf{S}}) = f(x_{\mathsf{C}} + x_{\mathsf{S}})$ for some operation $\cdot$ on the f co-domain, but such a homomorphism could also speed-up the ODA. For example, function $f(x) = g^x$ where $\mathbb{G} = \langle g \rangle$ is a multiplicative group of prime order p, satisfies the above if $\mathcal{X} = \mathbb{Z}_p$, but given $F = g^x$ one can find x in $\mathcal{O}(|\mathcal{X}|^{1/2})$ time using Shank's algorithm, which finds a collision between values $(g^{2^n})^{x_L}$ and $F \cdot g^{-x_R}$ for all n-bit x_L and x_R and $n = \lceil |p|/2 \rceil$.

We initiate the study of tight OWF's with algebraic properties that allow for efficient secure computation of f evaluation and/or equality-checking, on secret-shared inputs. We propose two such functions:

$$f_{\mathsf{Vand}}(x) = \prod_{i=1}^{N} g_i^{(x^i)} \qquad\qquad f_{\mathsf{PoNR}}(x) = \prod_{i=1}^{N} g_i^{(\prod_{x[j]=1} K_{i,j})}$$

where N is a 'repetition parameter', g_i's are elements of group $\mathbb{G}$ that are part of the function description, and (in f_{PoNR}) $K_{i,j}$'s are *public* random elements in $\mathbb{Z}_p$ that are also part of the function description. We call the first function 'Vandermonde' because the Vandermonde matrix comes up in its analysis, and we call the second function a 'product of Naor-Reingold PRFs' [39].

We show that in the Generic Group Model (GGM) both functions realize a UC notion $\mathcal{F}_{\mathsf{OWF}}$ of a *tight* OWF, which implies that for any domain subset S, it takes $\Omega(|S|)$ time to recover x given $f(x)$. Consequently, if (pw, s) is random in $D \times \{0,1\}^t$ then it takes $\Omega(|D| \cdot 2^t)$ time to recover pw from $f(\mathsf{pw}, s)$. In more detail, we say that f realizes $\mathcal{F}_{\mathsf{OWF}}$ with *rate* r if an n-time attacker can effectively test if $F = f(x^*)$ on at most $\mathcal{O}(n^r)$ candidates x^*, and we call f *tight* if it realizes $\mathcal{F}_{\mathsf{OWF}}$ with rate 1. We show that both functions f_{Vand} and f_{PoNR} realize $\mathcal{F}_{\mathsf{OWF}}$ with rate $r \leq 1 + d/N$ where $d = 2$ for f_{Vand} and $d \approx 3.3$ for f_{PoNR}, and with rate 1 if $N = |p|$. We note that even though these are the best bounds we could prove, either function *might* realize $\mathcal{F}_{\mathsf{OWF}}$ with rate 1 even for $N = 2$.

Contribution IV: CKEMs for Tight OWF Candidates. Both functions f_{Vand} and f_{PoNR} enable efficient realizations of CKEM for condition SSEM.

To simplify this technical overview assume $x = \mathsf{pw}|s$ is additively secret-shared as $x = x_{\mathsf{C}} + x_{\mathsf{S}} \bmod p$ between client C and server S. The Conditional KEM for the condition that S-held F satisfies $F = f_{\mathsf{Vand}}(x)$ can proceed by computing group element $F' = f_{\mathsf{Vand}}(x_{\mathsf{C}} + x_{\mathsf{S}})$ under an encryption which is homomorphic over group $\mathbb{G}$, i.e. ElGamal (the algebraic structure of f_{Vand} allows for this 2-party computation using $\mathcal{O}(N)$ group elements), and then folding key establishment into a test for plaintext equality of two ElGamal ciphertexts, c and c', based on the fact that c/c' raised to a random power encrypts 1 iff $F = F'$.

[9] In the explanation below we simplify notation by assuming that the whole argument x is secret-shared between C and S, whereas our actual TOWF proposals split the argument into pw and s and treat them differently since only s is secret-shared.

If F-commitment is implemented with the same ElGamal encryption then it is not hard to add a verification that the server runs the CKEM on committed F.

If x is an n-bit string which is xor-shared, i.e. $x = x_{\mathsf{C}} \oplus x_{\mathsf{S}}$, then the CKEM for the condition that F satisfies $F = f_{\mathsf{PoNR}}(x)$ can be similarly done by computing $F' = f_{\mathsf{PoNR}}(x_{\mathsf{C}} \oplus x_{\mathsf{S}})$ under ElGamal encryption, but this time we use a 2-party computation similar to the Oblivious PRF protocol for the Naor-Reingold (NR) PRF given by Freedman et al. [22]: Their OPRF protocol lets e.g. S compute the NR PRF $f_K(x)$ on S-held x and C-held key K, using $n = |x|$ Oblivious Transfer (OT) instances, where S uses the i-th bit of x on the i-th OT instance. In our case x is xor-shared between C and S (and key K is public!), but the OPRF of [22] is almost the same if x is xor-shared between the two parties: S still uses the i-th bit of its input x_{S} as the choice bit in the i-th OT, but C flips its OT inputs according to the i-th bit of its share x_{C}.

Relation to Existing (a)PAKEs. We follow a long line of works on PAKEs, e.g. [5,4,9,26,34,1,13,30], and aPAKEs, e.g. [31,25,6,32,10,29,18]. Typical (a)PAKEs, however, consider only 2 parties, e.g. client C and server S, while the TFA-KE setting considers device D as an additional 3rd party. Our ODA-strengthened TFA-KE most closely compares to *threshold* PAKE [21,36], which in the 2-server setting shares password information between two servers s.t. compromise of either one leaks no information and hence prevents an ODA attack. If device D acted as threshold server S_1 and server S as S_2, we would have a TFA with perfect ODA security upon compromise of S. However, existing 2-server threshold PAKE solutions assume that all parties can communicate freely, while in our setting $\mathsf{S}_1 = \mathsf{D}$ has severe bandwidth constraints. Indeed, our TFA-KE can be considered as a novel type of threshold *asymmetric PAKE* [28], which shares a password hash between $\mathsf{S}_1 = \mathsf{D}$ and $\mathsf{S}_2 = \mathsf{S}$, but where S_1 is allowed to send only *a single t-bit message*, on a secure channel to the client C, and where in such a setting the upper-bound on the ODA upon compromise of S_2 is $\mathcal{O}(|D| \cdot 2^t)$.

On Modularity, Composability, and the Generic Group Model. As explained above, we modularize our TFA-KE constructions into two building-blocks: a tight OWF and a corresponding CKEM. We do so for three reasons: (1) To present each building-block more clearly in isolation, because they are individually complex; (2) To highlight the generality of our approach and how it can be instantiated with different building-blocks; and (3) To minimize our dependence on the GGM. The tightness of our OWF's is proven using the GGM, but the security of our CKEM's is not, which formalizes the intuition that *our TFA-KE protocols rely on the GGM only for their security against ODA*.

Still, the combination of the GGM with the universal composability model raises some questions. In particular, we recognize that our tight OWF's may not compose with other protocols that use the same group and also model it as generic in their security arguments. This deficiency is substantially mitigated by our use of a *multi-session* ideal OWF functionality, which at least implies that it is safe for all OWF instances to use the same group. (Likewise our ideal TFA-KE functionality, and the analysis of a TFA-KE scheme, uses a multi-session setting which considers any number of TFA-KE instances, involving any number of

clients, servers, and password accounts at any given server.) We also emphasize that our way of modeling OWF's has merits regardless of the implications for composability. For example, it affords us a clean and natural way to reason about tightness as a property related to simulation rate.

Extensions and Questions. Our results open up several interesting questions and directions for future work. For example, aPAKE has been extended to *strong/salted* aPAKE [32] to prevent offline attacks from being precomputed before the time of server compromise. To similarly upgrade our TFA-KE protocols we would require a salted tight OWF with an analogous property. Gu et al. [28] showed how to use a threshold Oblivious PRF (OPRF) to implement a threshold aPAKE, where the ODA attack requires compromising t 'auxilliary servers' in addition to the 'target server'. For TFA-KE, a similar threshold OPRF technique should achieve a corresponding upgrade to the TFA-KE security guarantees. Another attractive TFA-KE improvement would be *forward security* of codes. For example, if the server were to 'ratchet' their code-generating PRF key k after each session by hashing it and deleting the old value, then the leakage of code for time T (i.e. lunch-time attack) would not help an offline attacker who compromises the server at time $T + 1$ or later.

There is also significant room for further exploration in the study of tight OWFs with algebraic properties. Our analyses leave a gap between the proven tightness of our proposed functions and the best-known explicit attacks against them. There is also the opportunity to propose and consider entirely new tight OWF candidates with the requisite properties that enable efficient 2-party secure evaluation or equality-checking on secret-shared inputs.

Note on Efficiency. We note that the TFA-KE protocols we present in this paper are much less efficient than the most efficient aPAKEs, e.g. [25,29,18] or even *strong* aPAKEs, e.g. [32,10], which all use the same prime-order groups and communicate $\mathcal{O}(1)$ group elements and perform $\mathcal{O}(1)$ group exponentiations. The cost of our TFA-KE based on TOWF candidate f_{Vand}, which is probably less expensive than the scheme based on f_{PoNR}, is dominated by Zero-Knowledge proofs of $\mathcal{O}(N+t)$ discrete logarithm relations. Even if we treated N and t as constants, the constants involved will be much larger than in the cited aPAKE's. Moreover, our analysis requires these proofs to be straight-line extractable (and UC). Using Fischlin's proofs [20], as recently optimized by Chen and Lindell [14], this would add a multiplicative factor of about 15x slow-down to an implementation. This count also omits an integer range ZK proof used in our scheme, but with the recent range proof techniques, see e.g. [15], the range proof cost would be insignificant compared to the rest. As for our TFA-KE using f_{PoNR} its cost will in addition require Committed OT's which implemented e.g. as in [24,33] would require $\mathcal{O}(N \cdot t)$ composite-order exponentiations.

2 Preliminaries

Hardness Assumptions. We defer the definitions of the Discrete Logarithm (DL) and Decisional Diffie-Hellman (DDH) assumptions to Appendix A.

Commitment Schemes. A commitment scheme $\mathsf{COM} : \mathcal{M} \times \mathcal{D} \to \mathcal{C}$ is an algorithm that on message $m \in \mathcal{M}$ and randomness $d \in \mathcal{D}$, outputs a commitment $C \in \mathcal{C}$. We do not define a separate algorithm for decommitment verification. To decommit a commitment C, the committer simply reveals m, d, and the verifier checks whether $\mathsf{COM}(m; d) = C$. A commitment scheme is required to be hiding and binding, and we defer the formal definitions of these two properties to Appendix A. When a commitment scheme is used with randomly sampled $d \leftarrow_{\$} \mathcal{D}$ (as is typical), we will sometimes write d as an output of COM rather than an input for notational convenience.

ElGamal and Pedersen Commitments. For commiting to messages in $\mathcal{M} = \mathbb{G}$, we use the ElGamal commitment scheme denoted $\mathsf{ElG.Commit}$, i.e. ElGamal encryption under a random public key. For commiting to messages in $\mathcal{M} = \mathbb{Z}_p^N$, we use the (compacted) Pedersen commitment scheme denoted Ped^N. We include formal definitions of $\mathsf{ElG.Commit}$ and Ped^N in Appendix A. Where the meaning is clear from context, we will sometimes abbreviate $\mathsf{ElG.Commit}$ as simply ElG.

Non-Interactive Zero-Knowledge Proofs. Figure 1 is $\mathcal{F}_{\mathsf{NIZK}}$, the ideal functionality for a non-interactive zero-knowledge proof (NIZK) system. The functionality is parameterized by relation $\mathcal{R}$, which is a set of statement-witness pairs. Our protocols use $\mathcal{F}_{\mathsf{NIZK}}$ for various relations, and we discuss in Appendix F that all of these NIZKs have efficient realizations from standard techniques.

- On $(\textsc{Prove}, \mathsf{ssid}, \mathbb{x}, \mathbb{w})$ from party P: If $\mathcal{R}(\mathbb{x}, \mathbb{w}) = 1$ then send $(\textsc{Prove}, \mathsf{P}, \mathsf{ssid}, \mathbb{x})$ to $\mathcal{A}^*$, await response $(\textsc{Proof}, \mathsf{ssid}, \pi)$ from $\mathcal{A}^*$, save record $(\mathbb{x}, \mathbb{w}, \pi)$, and send $(\textsc{Proof}, \mathsf{ssid}, \pi)$ to P.
- On $(\textsc{Verify}, \mathsf{ssid}, \mathbb{x}, \pi)$ from party V: If there exists no record $(\mathbb{x}, \mathbb{w}, \pi)$ then send $(\textsc{Verify}, \mathsf{V}, \mathsf{ssid}, \mathbb{x}, \pi)$ to $\mathcal{A}^*$, await response $(\textsc{Witness}, \mathsf{ssid}, \mathbb{w})$, and save record $(\mathbb{x}, \mathbb{w}, \pi)$. In either case, send $(\textsc{Verification}, \mathsf{ssid}, \mathcal{R}(\mathbb{x}, \mathbb{w}))$ to V.

Fig. 1. $\mathcal{F}_{\mathsf{NIZK}}[\mathcal{R}]$: Non-interactive zero-knowledge proof functionality

Committed Oblivious Transfer. Introduced in [17], committed oblivious transfer (COT) is a strengthened variant of standard oblivious transfer that allows parties to prove correct behavior relative to committed values. We defer a full definition and discussion of COT to Appendix A.

3 Tight One-Way Functions

Figure 2 is $\mathcal{F}_{\mathsf{OWF}}$, an ideal functionality for a one-way function (OWF) with domain $\mathcal{X}$. This functionality allows honest parties to evaluate the OWF at will,

with each evaluation tagged by a session identifier sid. Once a function output is leaked to an attacker, they are able to test guessed inputs against it and learn whether or not their guesses are correct.

Messages with $x, x^* \notin \mathcal{X}$ are ignored by the functionality.

- On $(\textsc{Eval}, \mathsf{sid}, x)$ from any honest party P: if this is the first such message for sid, record $(\mathsf{sid}, \mathsf{P}, x)$ marked $\mathsf{uncompromised}$, and send $(\textsc{Eval}, \mathsf{sid}, \mathsf{P})$ to $\mathcal{A}^*$.
- On $(\textsc{Test}, \mathsf{sid}, x^*)$ from P where there exists record $(\mathsf{sid}, \mathsf{P}, x)$: if $x^* = x$, return "correct"; otherwise return "wrong".

- On $(\textsc{LeakVal}, \mathsf{sid})$ from $\mathcal{A}^*$: mark record $(\mathsf{sid}, \mathsf{P}, x)$ $\mathsf{compromised}$ (if it exists).
- On $(\textsc{AdvTest}, \mathsf{sid}, x^*)$ from $\mathcal{A}^*$ where there exists record $(\mathsf{sid}, \mathsf{P}, x)$ marked $\mathsf{compromised}$: if $x^* = x$, return "correct"; otherwise return "wrong".

Fig. 2. $\mathcal{F}_{\mathsf{OWF}}[\mathcal{X}]$: One-way function functionality, parameterized by domain $\mathcal{X}$

Crucially, all adversarial function evaluations must be performed via interaction with the ideal functionality. Therefore, $\mathcal{F}_{\mathsf{OWF}}$ is only realizable in a non-plain model of computation such as the Random Oracle Model (ROM) or Generic Group Model (GGM). In these models, it is assumed that there exist computations that cannot be performed "locally" without interacting with an ideal oracle. In order to invert the OWF, then, a real-world attacker must perform some such interactions (e.g. random oracle queries or generic group operations), which cause activation of the simulator SIM. If the OWF realizes $\mathcal{F}_{\mathsf{OWF}}$, there must exist a SIM that can answer these oracle interactions while performing at most a polynomial number of queries to $\textsc{AdvTest}$. The smaller this polynomial's degree, the *tighter* the OWF, i.e. the more computational work the attacker must perform in order to invert the function.

Definition 1 (OWF Tightness). *OWF simulator* SIM *has* rate r *if it performs at most* $\mathcal{O}(n^r)$ *queries to* $\mathcal{F}_{\mathsf{OWF}}$ *after* n *activations. Similarly, OWF* f *has* tightness r *if it realizes* $\mathcal{F}_{\mathsf{OWF}}$ *with a simulator of rate* r.[10]

We will say a OWF is *tight* if it has tightness $r = 1$, which is optimal. In this case, there is no way of inverting the function that is faster (up to a constant factor) than a naive brute-force attack.

Comparison to [10]. Our $\mathcal{F}_{\mathsf{OWF}}$ functionality is based on the similar functionality given in [10], but with two key additions.

First, we explicitly model a $\textsc{LeakVal}$ message in the functionality, rather than leave this leakage as an implicit prerequisite of the $\textsc{AdvTest}$ message.

[10] Some previous works [37,38] have instead defined simulation rate as a function $r(n)$ s.t. simulator SIM has rate $r(n)$ if it performs at most $n \cdot r(n)$ functionality queries after n activations.

Looking ahead, we will require our simulator to simulate leaked OWF outputs for the adversary, so it is helpful for the event to be modeled clearly.

Second, we add the TEST command to the honest parties' interface. Without it, $\mathcal{F}_{\mathsf{OWF}}$ could be realized trivially, e.g. by the constant function $f(\mathsf{sid}, x) = 0$, which can be simulated without ever querying ADVTEST. To avert such possibilities, [10] augment their ideal functionality with a separately defined *collision-resistance* property. Our TEST interface achieves the same effect: if the environment finds two inputs x_1, x_2 which map to the same OWF output, then $(\text{EVAL}, \mathsf{sid}, x_1)$ followed by $(\text{TEST}, \mathsf{sid}, x_2)$ will return "correct" in the real world but "wrong" in the ideal world. Thus, our functionality actually models a *collision-resistant* OWF. Looking ahead, this stronger property is indeed needed by our TFA-KE application, to ensure that each online session corresponds to just a single $(\mathsf{pw}, \mathsf{code})$ guess from each party.

Example of a Tight OWF. In the ROM, our tight OWF definition has an immediate realization: $f_{\mathsf{RO}}(\mathsf{sid}, x) := \mathsf{H}(\mathsf{sid}, x)$. To see this, consider simulator SIM which, in the epoch before $(\text{LEAKVAL}, \mathsf{sid})$ occurs, simply samples random $\mathsf{H}(\mathsf{sid}, x^*)$ values. Upon $(\text{LEAKVAL}, \mathsf{sid})$, SIM queries $(\text{ADVTEST}, \mathsf{sid}, x^*)$ for each previously sent x^*. If one of them x returns "correct", then SIM sends the previously sampled $\mathsf{H}(\mathsf{sid}, x)$ as the leaked value; otherwise, SIM samples and sends a new random value y. On each subsequent $\mathsf{H}(\mathsf{sid}, x^*)$ call, SIM queries $(\text{ADVTEST}, \mathsf{sid}, x^*)$ and programs $\mathsf{H}(\mathsf{sid}, x^*) := y$ if it returns "correct"; otherwise, SIM continues to sample random values. SIM performs exactly 1 ADVTEST query per H activation; thus, f_{RO} is tight.

Unfortunately, this simple OWF is not amenable to our TFA application. Looking ahead to Section 5, we will require an online OWF evaluation protocol that runs on a function input $x = (\mathsf{pw}, s)$ that is secret-shared between the client and server. (Specifically, s is secret-shared and pw is held by the client.) Since neither party can reveal their share, use of a black-box hash function seems to require generic MPC techniques for such evaluation, which would impose major efficiency costs. Our challenge is to find tight OWFs with an algebraic structure that enables more efficient function computation on secret-shared inputs.[11]

3.1 Tight OWF Candidate: Function f_{Vand}

We propose the following function $f_{\mathsf{Vand}} : \{0,1\}^* \times \{0,1\}^* \times \{0,1\}^t \to \mathbb{G}$ as a tight OWF.[12] Tailored to our TFA-KE use case, we treat function inputs as tuples $(\mathsf{sid}, \mathsf{pw}, s)$ where sid is the session identifier, pw is a password, and s is a t-length bitstring. f_{Vand} is parameterized by group $\mathbb{G}$ of prime order p,

[11] The tight OWF proposed by [10] does not have this structure. That work seeks a tight OWF with an algebraic structure that enables efficient computation between two parties, one holding a high-entropy salt and the other a low-entropy function input. The Boneh-Boyen OWF they utilize is only tight when function inputs are hashed, and it is therefore unsuited to our case when they are secret-shared.

[12] The name of f_{Vand} refers to the Vandermonde matrix that arises in its analysis.

a "repetition number" $N \in \mathbb{Z}^+$, and hash functions $\mathsf{H}_{\mathbb{G}} : \{0,1\}^* \to \mathbb{G}$ and $\mathsf{H}_{\mathsf{p}} : \{0,1\}^* \to \{0,1\}^{\lfloor \log p \rfloor - t - 1}$.

Definition 2 (f_{Vand}).

$$f_{\mathsf{Vand}}(\mathsf{sid}, \mathsf{pw}, s) := \prod_{i \in [N]} \mathsf{H}_{\mathbb{G}}(\mathsf{sid}, i)^{k_i(\mathsf{pw}, s)}$$

$$\text{where} \qquad k_i(\mathsf{pw}, s) := (\mathsf{H}_{\mathsf{p}}(\mathsf{pw}) | 0 | s)^i$$

Here, $(\mathsf{H}_{\mathsf{p}}(\mathsf{pw})|0|s)$ is the $\mathbb{Z}_p$ element represented by the concatenation of bitstrings $\mathsf{H}_{\mathsf{p}}(\mathsf{pw})$, 0, and s.

We will find it useful in our analysis to decompose f_{Vand} into a pre-processing step f_{pre} and a main step f^*_{Vand}.

Definition 3 (f_{pre}).

$$f^{\mathcal{HP}}_{\mathsf{pre}} : \{0,1\}^* \times \{0,1\}^* \times \{0,1\}^t \to \{0,1\}^* \times \mathcal{HP} \times \{0,1\}^t$$
$$f^{\mathcal{HP}}_{\mathsf{pre}}(\mathsf{sid}, \mathsf{pw}, s) := (\mathsf{sid}, \mathsf{H}_{\mathsf{p}}(\mathsf{pw}), s) \quad \text{where} \quad \mathsf{H}_{\mathsf{p}} : \{0,1\}^* \to \mathcal{HP}$$

Defining $f^*_{\mathsf{Vand}}(\mathsf{sid}, \mathsf{hpw}, s) := \prod_{i \in [N]} \mathsf{H}_{\mathbb{G}}(\mathsf{sid}, i)^{(\mathsf{hpw}|0|s)^i}$, we have $f_{\mathsf{Vand}} = f^*_{\mathsf{Vand}} \circ f^{\mathcal{HP}}_{\mathsf{pre}}$ for $\mathcal{HP} = \{0,1\}^{\lfloor \log p \rfloor - t - 1}$.

Theorem 1. *Function f_{Vand} realizes $\mathcal{F}_{\mathsf{OWF}}[\{0,1\}^* \times \{0,1\}^t]$, assuming hash function $\mathsf{H}_{\mathbb{G}}$ is modeled as a random oracle, hash function H_{p} is modeled as a non-programmable random oracle, $\mathbb{G}$ is modeled as a generic group, at most $q_{\mathsf{H}} \leq 2^{(\lfloor \log p \rfloor - t - 1 - \kappa)/2}$ queries are made to H_{p}, at most $q_{\mathbb{G}}$ $\mathbb{G}$ elements are seen by the environment, and $q_{\mathsf{H}} {q_{\mathbb{G}}}^2 \leq 2^{\lfloor \log p \rfloor - t - 1 - \kappa}$.*

Proof. To prove the security of f_{Vand} we provide in Figure 3 the simulator $\mathsf{SIM}_{\mathsf{Vand}}$, which operates in the GGM and ROM. In particular, $\mathsf{SIM}_{\mathsf{Vand}}$ must appropriately program the responses to every $\mathsf{H}_{\mathbb{G}}$ evaluation and $\mathbb{G}$ group operation while making a minimal number of AdvTest queries to $\mathcal{F}_{\mathsf{OWF}}$. For $\mathsf{H}_{\mathbb{G}}$, $\mathsf{SIM}_{\mathsf{Vand}}$ simply samples responses at random.

For group operations, the simulation strategy is more complex. $\mathsf{SIM}_{\mathsf{Vand}}$ maintains an internal description of every $\mathbb{G}$ element the environment has seen and constantly checks for pairs of these group elements that represent tests for possible OWF inversions (Condition 2). If two group elements do represent a test, then $\mathsf{SIM}_{\mathsf{Vand}}$ performs the relevant AdvTest queries to evaluate it. If the test succeeds, then the simulator correctly programs the two associated $\mathbb{G}$ elements to be identical. Notice that a "test pair" for sid is only possible after the (LeakVal, sid) event; before that event, $\beta[\gamma](\mathsf{sid}) = 0$ for all γ.

Thus, $\mathsf{SIM}_{\mathsf{Vand}}$ perfectly emulates the real world, except in the case of three "bad events" which we argue to be negligibly probable.

1. Discrete Logarithm break. $\mathsf{SIM}_{\mathsf{Vand}}$ does not bother to define the discrete logarithms between $\mathbb{G}$ elements output by $\mathsf{H}_{\mathbb{G}}$. The only relationship between $\mathbb{G}$ elements that is respected by $\mathsf{SIM}_{\mathsf{Vand}}$ is that of the OWF f_{Vand}, which

Whenever a new $\mathbb{G}$ element is sent to the environment, the simulator picks a unique random string to represent it (except when this behavior is overridden by the code below). The simulator maintains internal descriptions for these $\mathbb{G}$ elements in the form of two tables of maps α and β. $\alpha[\gamma]$ is a map $(\mathsf{sid}, i) \mapsto \mathbb{Z}_p$ and $\beta[\gamma]$ is a map $\mathsf{sid} \mapsto \mathbb{Z}_p$ such that $\gamma \in \mathbb{G}$ is described as:

$$\gamma = \prod_{\mathsf{sid},i} \mathsf{H}_{\mathbb{G}}(\mathsf{sid}, i)^{\alpha[\gamma](\mathsf{sid},i)} \cdot \prod_{\mathsf{sid}} (y_{\mathsf{sid}})^{\beta[\gamma](\mathsf{sid})} \tag{1}$$

The simulator also maintains a table $X : (\mathsf{sid}, \mathsf{pw}, s) \mapsto \{0, 1\}$ of possible OWF inversions that are known to be either correct or wrong.

- On fresh query $\mathsf{H}_{\mathbb{G}}(\mathsf{sid}, i)$, define new $\gamma \in \mathbb{G}$ and return it. $\alpha[\gamma](\mathsf{sid}, i) := 1$.
- On (LeakVal, sid), define new $y_{\mathsf{sid}} \in \mathbb{G}$ and send it to $\mathcal{A}^*$. $\beta[y_{\mathsf{sid}}](\mathsf{sid}) := 1$.
- On group operation $\gamma_1 \cdot \gamma_2$, define new $\mathbb{G}$ element γ_3 and return it. $\alpha[\gamma_3] := \alpha[\gamma_1] + \alpha[\gamma_2]$ and $\beta[\gamma_3] := \beta[\gamma_1] + \beta[\gamma_2]$.

In the above, unspecified values of the α and β maps default to 0, and $+$ represents component-wise addition of the full maps, including all sid's.

Whenever a new $\mathbb{G}$ element γ is defined (by $\mathsf{H}_{\mathbb{G}}$ query, LeakVal event, or group operation), the simulator checks whether its internal description is identical to any previous γ'; if so, the same unique string is used as its representation. If not, the simulator proceeds to check whether the internal description of γ is *conditionally* equivalent to any previous γ', conditioned on possible OWF inversions:

$$\begin{aligned}
&\forall_{\mathsf{sid}\ \text{s.t.}\ \beta[\gamma](\mathsf{sid})=\beta[\gamma'](\mathsf{sid})} \\
&\quad \forall_{i\in[N]} \quad \alpha[\gamma](\mathsf{sid}, i) = \alpha[\gamma'](\mathsf{sid}, i) \\
&\qquad\qquad \wedge \\
&\forall_{\mathsf{sid}\ \text{s.t.}\ \beta[\gamma](\mathsf{sid})\neq\beta[\gamma'](\mathsf{sid})} \quad \exists_{\mathsf{pw},s} \\
&\quad \forall_{i\in[N]} \quad \alpha[\gamma](\mathsf{sid}, i) - \alpha[\gamma'](\mathsf{sid}, i) = k_i(\mathsf{pw}, s) \cdot (\beta[\gamma](\mathsf{sid}) - \beta[\gamma'](\mathsf{sid}))
\end{aligned} \tag{2}$$

If Condition 2 is true with some $(\mathsf{sid}_1, \mathsf{pw}_1, s_1), (\mathsf{sid}_2, \mathsf{pw}_2, s_2), \ldots, (\mathsf{sid}_n, \mathsf{pw}_n, s_n)$ satisfying the second clause, then consider each of those as a possible OWF inversion. If all these inversions are correct, then $\gamma = \gamma'$; otherwise, $\gamma \neq \gamma'$. If there is insufficient information in X to determine whether $\gamma = \gamma'$, perform the relevant (AdvTest, sid, (pw, s)) queries until receiving one "wrong" or receiving all "correct". Record responses in X to avoid redundant queries in the future.

Fig. 3. $\mathsf{SIM}_{\mathsf{Vand}}$: Simulator for f_{Vand}

is checked in Condition 2. In the real world, it is possible for the environment to compute a discrete logarithm relation between $\mathsf{H}_{\mathbb{G}}$ responses and thereby find alternative descriptions for group elements. In the ideal world, $\mathsf{SIM}_{\mathsf{Vand}}$ will erroneously program these alternative descriptions as if they describe unique group elements.

The classical GGM result of Shoup [43] is easily extended to show that this event occurs with probability $< {q_{\mathbb{G}}}^2/p$. Under the assumption that ${q_{\mathbb{G}}}^2 < 2^{\log p - \kappa}$, this probability is $< 2^{-\kappa}$.

2. Hash Premonition. When checking Condition 2, $\mathsf{SIM}_{\mathsf{Vand}}$ only considers pw's that have been queried to H_{p}. It is possible for a later H_{p} query to sample a $\mathsf{hpw} \leftarrow_{\$} \mathcal{HP} = \{0,1\}^{\lfloor \log p \rfloor - t - 1}$ that retroactively satisfies Condition 2 for a pair γ, γ' of $\mathbb{G}$ elements that $\mathsf{SIM}_{\mathsf{Vand}}$ already judged to be unequal.

For every $\gamma, \gamma' \in \mathbb{G}$ and every sid s.t. $\beta[\gamma](\mathsf{sid}) \neq \beta[\gamma'](\mathsf{sid})$, there is a single $\mathsf{hpw} \in \mathcal{HP}$ that satisfies Condition 2. Thus, the probability that a new H_{p} query retroactively satisfies Condition 1 for any γ, γ' is $\leq {q_{\mathbb{G}}}^2/|\mathcal{HP}|$. The probability of the event across q_{H} H_{p} queries is $\leq q_{\mathsf{H}} {q_{\mathbb{G}}}^2/|\mathcal{HP}|$. Under the assumption that $q_{\mathsf{H}} {q_{\mathbb{G}}}^2 \leq 2^{\lfloor \log p \rfloor - t - 1 - \kappa}$, this probability is $\leq 2^{-\kappa}$.

3. OWF Collision. As mentioned above, $\mathcal{F}_{\mathsf{OWF}}$ requires that a OWF be collision-resistant: If the environment can find $x_1 \neq x_2$ and sid s.t. $f(\mathsf{sid}, x_1) = f(\mathsf{sid}, x_2)$, then it is easy to distinguish the real and ideal worlds via the TEST command. This collision-resistance is a property of the OWF itself, i.e. it is independent of the simulator.

Lemma 1. *Function f^*_{Vand} is collision-resistant, assuming $\mathsf{H}_{\mathbb{G}}$ is modeled as a random oracle and DL is hard in $\mathbb{G}$.*

To prove Lemma 1, suppose there exists algorithm $\mathcal{A}$ that outputs $(\mathsf{hpw}, s) \neq (\mathsf{hpw}', s')$ and sid s.t. $f^*_{\mathsf{Vand}}(\mathsf{sid}, \mathsf{hpw}, s) = f^*_{\mathsf{Vand}}(\mathsf{sid}, \mathsf{hpw}', s')$. The two evaluations use the same bases, but $\forall_{i \in [N]} (\mathsf{hpw}|0|s)^i \neq (\mathsf{hpw}'|0|s')^i$. Then there is a straightforward and well-known "DL relation" reduction (e.g. [35]) from $\mathcal{A}$ that solves DL in $\mathbb{G}$.

By Lemma 1, in the case that no DL break (i.e. bad event 1) occurs, $f_{\mathsf{Vand}} = f^*_{\mathsf{Vand}} \circ f_{\mathsf{pre}}$ collision is only possible in the case of f_{pre} collision. Under the assumption that $q_{\mathsf{H}} \leq 2^{(\lfloor \log p \rfloor - t - 1 - \kappa)/2}$, the probability of such a collision is $< 2^{-\kappa}$ by a simple birthday bound.

It is now proven that f_{Vand} realizes $\mathcal{F}_{\mathsf{OWF}}$, but its tightness still remains to be analyzed.

Theorem 2 (f_{Vand} Tightness). *Assume that at most $q_{\mathsf{H}} \leq 2^{(\lfloor \log p \rfloor - t - 1 - \kappa)/2}$ queries are made to H_{p} and at most $q_{\mathbb{G}}$ $\mathbb{G}$ elements are seen by the environment.*

- *If $N \geq \log q_{\mathbb{G}}$, then $\mathsf{SIM}_{\mathsf{Vand}}$ has rate $r = 1$, i.e. f_{Vand} is tight.*
- *If $N < \log q_{\mathbb{G}}$, then $\mathsf{SIM}_{\mathsf{Vand}}$ has rate $r = 1 + \frac{1}{\lfloor N/2 \rfloor}$.*

Proof. To begin, split the ADVTEST queries made by $\mathsf{SIM}_{\mathsf{Vand}}$ into those which return "correct" and those which return "wrong". Once a "correct" value is

found, no other queries are ever needed for that sid. Therefore, the total number of "correct" AdvTest queries is upper-bounded by the number of (LeakVal, sid) activations of $\mathsf{SIM}_{\mathsf{Vand}}$, i.e. the rate of such queries is 1.

To upper-bound the number of "wrong" queries, it is helpful to think of an *attack graph* $\mathcal{G}$. This undirected graph has at most $q_{\mathbb{G}}$ vertices, one for each $\mathbb{G}$ element seen by the environment. It has an edge between every pair of $\mathbb{G}$ elements that triggered a "wrong" test. Every edge is associated with the $(\mathsf{sid}, \mathsf{pw}, s)$ that it tests for. The simulator only makes at most a single "wrong" query for every pair; the attack graph is therefore simple and contains at most $\mathcal{O}(n^2)$ edges. Consequently, $\mathsf{SIM}_{\mathsf{Vand}}$ has rate $r \leq 2$ (regardless of parameter N). To improve this bound, we must bound the edge density of $\mathcal{G}$.

To understand the definition of the attack graph $\mathcal{G}$, it is helpful to consider an example. (For simplicity, we consider the case of a single sid, and we omit it from notation.) A typical adversary would first perform LeakVal and learn y; $\mathsf{SIM}_{\mathsf{Vand}}$ records $\beta[y] = 1$ and $\alpha[y](i) = 0$ for all $i \in [N]$. To test an inversion $y \stackrel{?}{=} f_{\mathsf{Vand}}(\mathsf{pw}_1, s_1)$, the adversary might then compute γ_1 with $\beta[\gamma_1] = 0$ and $\alpha[\gamma_1](i) = k_i(\mathsf{pw}_1, s_1)$ for all $i \in [N]$, i.e. $\gamma_1 = \prod_{i\in[N]} \mathsf{H}_{\mathbb{G}}(i)^{k_i(\mathsf{pw}_1,s_1)}$. Per Condition 2, y and γ_1 trigger an AdvTest for (pw_1, s_1) and therefore create an edge (y, γ_1) in $\mathcal{G}$. If the adversary repeats this behavior for other (pw, s) tests, $\mathcal{G}$ will be a star graph with y as its center. However, other adversarial behaviors are also possible. For instance, suppose the adversary computes γ_2 with $\beta[\gamma_2] = 7$ and $\alpha[\gamma_2](i) = k_i(\mathsf{pw}_1, s_1) - 7k_i(\mathsf{pw}_2, s_2)$ for all $i \in [N]$, i.e. $\gamma_2 = (y)^7 \cdot \prod_{i\in[N]} \mathsf{H}_{\mathbb{G}}(i)^{k_i(\mathsf{pw}_1,s_1)-7k_i(\mathsf{pw}_2,s_2)}$. Now γ_1 and γ_2 constitute an AdvTest for (pw_2, s_2) and therefore create an edge (γ_1, γ_2) in $\mathcal{G}$. In the face of arbitrary adversarial behavior, $\mathcal{G}$ could become much more complex than a star graph.

Toward bounding the edge density of $\mathcal{G}$, we define $K((\mathsf{pw}_1, s_1), \ldots, (\mathsf{pw}_c, s_c))$ to be an $N \times c$ matrix that is a function of $(\mathsf{pw}_1, s_1), \ldots, (\mathsf{pw}_c, s_c) \in \{0,1\}^* \times \{0,1\}^t$ (for any $c \in \mathbb{Z}^+$):

$$K((\mathsf{pw}_1, s_1), \ldots, (\mathsf{pw}_c, s_c)) := \begin{bmatrix} k_1(\mathsf{pw}_1, s_1) & k_1(\mathsf{pw}_2, s_2) & \ldots & k_1(\mathsf{pw}_c, s_c) \\ k_2(\mathsf{pw}_1, s_1) & k_2(\mathsf{pw}_2, s_2) & \ldots & k_2(\mathsf{pw}_c, s_c) \\ \vdots & & \ddots & \vdots \\ k_N(\mathsf{pw}_1, s_1) & k_N(\mathsf{pw}_2, s_2) & \ldots & k_N(\mathsf{pw}_c, s_c) \end{bmatrix}$$

Using this notation, we take a step toward proving Theorem 2 with the following lemma.

Lemma 2. *For any integer $c \geq 3$, if the* AdvTest *queries of* $\mathsf{SIM}_{\mathsf{Vand}}$ *imply an attack graph $\mathcal{G}$ containing a cycle of length $\leq c$, then there exist distinct* (pw_1, s_1), *...,* $(\mathsf{pw}_c, s_c) \in \{0,1\}^* \times \{0,1\}^t$ *s.t.* $\mathsf{rank}(K((\mathsf{pw}_1, s_1), \ldots, (\mathsf{pw}_c, s_c))) < c$.

Suppose that $\mathcal{G}$ contains a cycle $C = (\gamma_1, \gamma_2, \ldots, \gamma_{\tilde{c}})$ of length $\tilde{c} \leq c$. Label it such that $(\gamma_1, \gamma_{\tilde{c}})$ is the first of C's edges added to $\mathcal{G}$ (i.e. the earliest of the corresponding AdvTest queries made by $\mathsf{SIM}_{\mathsf{Vand}}$), and consider the sid tested by that $(\gamma_1, \gamma_{\tilde{c}})$ edge. For every edge (γ, γ') in C, either $\forall_{i\in[N]}\ \alpha[\gamma](\mathsf{sid}, i) =$

$\alpha[\gamma'](\mathsf{sid}, i)$ or $\exists_{\mathsf{pw},s}\ \forall_{i\in[N]}\ \alpha[\gamma](\mathsf{sid}, i) - \alpha[\gamma'](\mathsf{sid}, i) = k_i(\mathsf{pw}, s) \cdot (\beta[\gamma](\mathsf{sid}) - \beta[\gamma'](\mathsf{sid})) \neq 0$ (by Condition 2). Define $\hat{C} = ((\hat{\gamma}_1, \hat{\gamma}'_1, \mathsf{pw}_1, s_1), (\hat{\gamma}_2, \hat{\gamma}'_2, \mathsf{pw}_2, s_2), \ldots, (\hat{\gamma}_{\hat{c}}, \hat{\gamma}'_{\hat{c}}, \mathsf{pw}_{\hat{c}}, s_{\hat{c}}))$ to be the subset of C's edges which meet the latter condition (and their associated pw, s values). $(\hat{\gamma}_1, \hat{\gamma}'_1) = (\gamma_1, \gamma_{\tilde{c}})$, so $1 \leq \hat{c} \leq \tilde{c} \leq c$.

For every $i \in [N]$, the sum of the difference of $\alpha[\cdot](\mathsf{sid}, i)$ across every edge in C must equal 0 (because C is a cycle). By definition, this difference is 0 for the edges not in $\hat{C}$. Therefore:

$$\begin{aligned}\forall_{i\in[N]} \qquad 0 &= \sum_{(\hat{\gamma},\hat{\gamma}',\mathsf{pw},s)\in\hat{C}} \alpha[\hat{\gamma}](\mathsf{sid}, i) - \alpha[\hat{\gamma}'](\mathsf{sid}, i) \\ &= \sum_{(\hat{\gamma},\hat{\gamma}',\mathsf{pw},s)\in\hat{C}} k_i(\mathsf{pw}, s) \cdot (\beta[\hat{\gamma}](\mathsf{sid}) - \beta[\hat{\gamma}'](\mathsf{sid}))\end{aligned}$$

Written in matrix form:

$$\begin{bmatrix}0\\0\\\vdots\\0\end{bmatrix} = \begin{bmatrix}k_1(\mathsf{pw}_1, s_1) & \ldots & k_1(\mathsf{pw}_{\hat{c}}, s_{\hat{c}})\\ k_2(\mathsf{pw}_1, s_1) & \ldots & k_2(\mathsf{pw}_{\hat{c}}, s_{\hat{c}})\\ \vdots & \ddots & \vdots\\ k_N(\mathsf{pw}_1, s_1) & \ldots & k_N(\mathsf{pw}_{\hat{c}}, s_{\hat{c}})\end{bmatrix} \cdot \begin{bmatrix}\beta[\hat{\gamma}_1](\mathsf{sid}) - \beta[\hat{\gamma}'_1](\mathsf{sid})\\ \beta[\hat{\gamma}_2](\mathsf{sid}) - \beta[\hat{\gamma}'_2](\mathsf{sid})\\ \vdots\\ \beta[\hat{\gamma}_{\hat{c}}](\mathsf{sid}) - \beta[\hat{\gamma}'_{\hat{c}}](\mathsf{sid})\end{bmatrix} \tag{3}$$

The $(\mathsf{pw}_1, s_1), \ldots, (\mathsf{pw}_{\hat{c}}, s_{\hat{c}})$ values are not necessarily distinct, which means that the matrix above may contain duplicate columns. However, it is guaranteed that (pw_1, s_1) is distinct from all $(\mathsf{pw}_2, s_2), \ldots, (\mathsf{pw}_{\hat{c}}, s_{\hat{c}})$. Recall that we labeled $(\gamma_1, \gamma_{\tilde{c}}) = (\hat{\gamma}_1, \hat{\gamma}'_1)$ such that it corresponds to the first AdvTest query of cycle C and that all the queries recorded in $\mathcal{G}$ are "wrong". Therefore, any future vertex pair whose Condition 2 implies a test for $(\mathsf{sid}, \mathsf{pw}_1, s_1)$ would not trigger an AdvTest query (i.e. would not add an edge to $\mathcal{G}$) because the recorded information $X[\mathsf{sid}, \mathsf{pw}_1, s_1] = 0$ is sufficient for $\mathsf{SIM}_{\mathsf{Vand}}$ to conclude that the $\mathbb{G}$ elements of the pair are not equal.

Define d to be the number of distinct (pw, s) in $(\mathsf{pw}_1, s_1), \ldots, (\mathsf{pw}_{\hat{c}}, s_{\hat{c}})$. Consider the condensed set $(\mathsf{pw}_{\rho_1}, s_{\rho_1}), \ldots, (\mathsf{pw}_{\rho_d}, s_{\rho_d})$, where $\rho_1, \ldots, \rho_d$ are the first indices of each distinct (pw, s) in $\hat{C}$, and $P_1, \ldots, P_d$ are the sets of all indices of each of them. $\rho_1 = 1$, $P_1 = \{1\}$, and $1 \leq d \leq \hat{c} \leq \tilde{c} \leq c$. Using this notation, we can now rewrite Equation 3 without duplicate columns:

$$\begin{bmatrix}0\\0\\\vdots\\0\end{bmatrix} = \begin{bmatrix}k_1(\mathsf{pw}_{\rho_1}, s_{\rho_1}) & \ldots & k_1(\mathsf{pw}_{\rho_d}, s_{\rho_d})\\ k_2(\mathsf{pw}_{\rho_1}, s_{\rho_1}) & \ldots & k_2(\mathsf{pw}_{\rho_d}, s_{\rho_d})\\ \vdots & \ddots & \vdots\\ k_N(\mathsf{pw}_{\rho_1}, s_{\rho_1}) & \ldots & k_N(\mathsf{pw}_{\rho_d}, s_{\rho_d})\end{bmatrix} \cdot \begin{bmatrix}\beta[\hat{\gamma}_1](\mathsf{sid}) - \beta[\hat{\gamma}'_1](\mathsf{sid})\\ \sum_{\rho\in P_2} \beta[\hat{\gamma}_\rho](\mathsf{sid}) - \beta[\hat{\gamma}'_\rho](\mathsf{sid})\\ \vdots\\ \sum_{\rho\in P_d} \beta[\hat{\gamma}_\rho](\mathsf{sid}) - \beta[\hat{\gamma}'_\rho](\mathsf{sid})\end{bmatrix}$$
$$\mathbf{0} = K((\mathsf{pw}_{\rho_1}, s_{\rho_1}), \ldots, (\mathsf{pw}_{\rho_d}, s_{\rho_d})) \cdot \boldsymbol{B} \tag{4}$$

Crucially, the topmost element of vector $\boldsymbol{B}$ must be non-zero (by the conditions given above), and therefore $\boldsymbol{B} = \mathbf{0}$ is not an acceptable solution to this system of equations. In order for the system to have a non-zero solution, the nullity of matrix K must be greater than 0, i.e. the rank of K must be less than d.

We conclude the proof of Lemma 2 by observing that if the $N \times d$ matrix $K((\mathsf{pw}_1, s_1), \ldots, (\mathsf{pw}_d, s_d))$ has rank $< d$, then an $N \times c$ matrix $K((\mathsf{pw}_1, s_1), \ldots, (\mathsf{pw}_d, s_d), (\mathsf{pw}_{d+1}, s_{d+1}), \ldots, (\mathsf{pw}_c, s_c))$ with rank $< c$ can be created by adding columns for any (distinct) $(\mathsf{pw}_{d+1}, s_{d+1}), \ldots, (\mathsf{pw}_c, s_c) \in \{0,1\}^* \times \{0,1\}^t$.

For $c > N$, all $N \times c$ matrices have rank $< c$ (in particular, they have rank $\leq N$). Consider, however, the case of $K((\mathsf{pw}_1, s_1), \ldots, (\mathsf{pw}_c, s_c))$ for $c \leq N$:

$$\begin{bmatrix} k_1(\mathsf{pw}_1, s_1) & \ldots & k_1(\mathsf{pw}_c, s_c) \\ k_2(\mathsf{pw}_1, s_1) & \ldots & k_2(\mathsf{pw}_c, s_c) \\ \vdots & \ddots & \vdots \\ k_N(\mathsf{pw}_1, s_1) & \ldots & k_N(\mathsf{pw}_c, s_c) \end{bmatrix} = \begin{bmatrix} (\mathsf{H_p}(\mathsf{pw}_1)|0|s_1)^1 & \ldots & (\mathsf{H_p}(\mathsf{pw}_c)|0|s_c)^1 \\ (\mathsf{H_p}(\mathsf{pw}_1)|0|s_1)^2 & \ldots & (\mathsf{H_p}(\mathsf{pw}_c)|0|s_c)^2 \\ \vdots & \ddots & \vdots \\ (\mathsf{H_p}(\mathsf{pw}_1)|0|s_1)^N & \ldots & (\mathsf{H_p}(\mathsf{pw}_c)|0|s_c)^N \end{bmatrix}$$

This is a Vandermonde matrix, and it has rank c iff all $(\mathsf{H_p}(\mathsf{pw}_1)|0|s_1), \ldots, (\mathsf{H_p}(\mathsf{pw}_c)|0|s_c)$ are distinct. Since (as argued in the proof of Theorem 1) an $\mathsf{H_p}$ collision is negligibly likely, it is sufficient that $(\mathsf{pw}_1, s_1), \ldots, (\mathsf{pw}_c, s_c)$ are distinct. We can therefore infer from Lemma 2 that the attack graph $\mathcal{G}$ of $\mathsf{SIM}_{\mathsf{Vand}}$ contains no cycles of length $\leq N$.

To complete the proof of Theorem 2, we refer to two theorems from extremal graph theory (cf. [23] for an excellent survey) that relate the lack of small cycles in $\mathcal{G}$ to its edge sparsity.

Theorem 3 ([2], [23, Theorem 4.1]). *For any integer $\lambda \geq 2$, if undirected graph $\mathcal{G}$ contains q vertices and no cycles of length $\leq \lambda$, then the number of edges in $\mathcal{G}$ is less than $\frac{1}{2}q^{1+1/\lfloor \lambda/2 \rfloor} + \frac{1}{2}q$.*

Theorem 4 ([19], [23, Theorem 4.39]). *For any integer $\delta \geq 2$, if undirected graph $\mathcal{G}$ contains q vertices and no cycles of length $< 2\log q / \log \delta$, then the number of edges in $\mathcal{G}$ is at most $\delta \cdot q$.*

Theorem 3 implies that the number of edges in $\mathcal{G}$ is $\mathcal{O}({q_{\mathbb{G}}}^{1+1/\lfloor N/2 \rfloor})$, i.e. $\mathsf{SIM}_{\mathsf{Vand}}$ has rate $r = 1 + \frac{1}{\lfloor N/2 \rfloor}$. Setting $\delta = 4$, Theorem 4 implies that if $N \geq \log q_{\mathbb{G}}$ then the number of edges in $\mathcal{G}$ is $\mathcal{O}(q_{\mathbb{G}})$, i.e. $\mathsf{SIM}_{\mathsf{Vand}}$ has rate $r = 1$ and f_{Vand} is tight. Together this implies Theorem 2.

3.2 Tight OWF Candidate: Function f_{PoNR}

Our second proposed tight OWF is $f_{\mathsf{PoNR}} : \{0,1\}^* \times \{0,1\}^* \times \{0,1\}^t \to \mathbb{G}$. Once again we treat function inputs as tuples $(\mathsf{sid}, \mathsf{pw}, s)$. f_{PoNR} is parameterized by group $\mathbb{G}$ of prime order p, repetition number $N \in \mathbb{Z}^+$, and hash functions $\mathsf{H}_{\mathbb{G}} : \{0,1\}^* \to \mathbb{G}$, $\hat{\mathsf{H}}_{\mathbb{G}} : \{0,1\}^* \to \mathbb{G}$, $\mathsf{H_p} : \{0,1\}^* \to \mathbb{Z}_p^*$, and $\mathsf{H_x} : \{0,1\}^* \to \mathbb{Z}_p^*$.

Definition 4 (f_{PoNR}).

$$f_{\mathsf{PoNR}}(\mathsf{sid}, \mathsf{pw}, s) := \prod_{i \in [N]} \mathsf{H}_{\mathbb{G}}(\mathsf{sid}, i)^{k_i(\mathsf{pw}, s)} \cdot \hat{\mathsf{H}}_{\mathbb{G}}(\mathsf{sid}, i)^{\mathsf{H_p}(i, \mathsf{pw})}$$

$$\text{where} \qquad k_i(\mathsf{pw}, s) := \mathsf{H_p}(i, \mathsf{pw}) \cdot \prod_{\substack{j \in [t] \\ s[j]=1}} \mathsf{H_x}(i, j)$$

This function follows the structure of f_{Vand} (Definition 2), differing only in the definition of $k_i(\mathsf{pw}, s)$ and the addition of the $\hat{\mathsf{H}}_{\mathbb{G}}$ bases (which are needed to ensure collision-resistance and are not involved in the proof of tightness).[13]

To aid analysis we decompose $f_{\mathsf{PoNR}} = f^*_{\mathsf{PoNR}} \circ f^{\mathcal{HP}}_{\mathsf{pre}}$ (Def. 3), where $\mathcal{HP} = (\mathbb{Z}^*_p)^N$ and $f^*_{\mathsf{PoNR}}(\mathsf{sid}, \mathsf{hpw}, s) := \prod_{i\in[N]} \mathsf{H}_{\mathbb{G}}(\mathsf{sid}, i)^{\mathsf{hpw}_i \cdot \prod_{j\in[t],s[j]=1} \mathsf{H}_{\mathsf{x}}(i,j)} \cdot \hat{\mathsf{H}}_{\mathbb{G}}(\mathsf{sid}, i)^{\mathsf{hpw}_i}$.

Lemma 3. *Function f^*_{PoNR} is collision-resistant, assuming $\mathsf{H}_{\mathbb{G}}$, $\hat{\mathsf{H}}_{\mathbb{G}}$, and H_{x} are modeled as random oracles and DL is hard in $\mathbb{G}$.*

Theorem 5. *Function f_{PoNR} realizes $\mathcal{F}_{\mathsf{OWF}}[\{0,1\}^* \times \{0,1\}^t]$, assuming hash functions $\mathsf{H}_{\mathbb{G}}$, $\hat{\mathsf{H}}_{\mathbb{G}}$, and H_{x} are modeled as random oracles, hash function H_{p} is modeled as a non-programmable random oracle, $\mathbb{G}$ is modeled as a generic group, at most $q_{\mathsf{H}} \leq 2^{(N \log p - \kappa)/2}$ queries are made to H_{p}, at most $q_{\mathbb{G}} \leq 2^{(\log p - \kappa)/2}$ $\mathbb{G}$ elements are seen by the environment, $q_{\mathsf{H}} {q_{\mathbb{G}}}^2 \leq 2^{N \log p - \kappa}$, and $N \geq \frac{\kappa + 2t}{\log(p-1)}$.*

Theorem 6 (f_{PoNR} Tightness). *Assume that at most $q_{\mathsf{H}} \leq 2^{(N \log p - \kappa)/2}$ queries are made to H_{p}, at most $q_{\mathbb{G}}$ $\mathbb{G}$ elements are seen by the environment, and $N \geq \frac{\kappa+2t}{\log(p-1)}$, and let c be the greatest integer s.t. $c - 1 + \frac{\kappa + c(t + \log q_{\mathsf{H}})}{\log(p-1) - \log(2c(t+1))} \leq N \leq 2c$.*

- *If $c \geq \log q_{\mathbb{G}}$, then $\mathsf{SIM}_{\mathsf{PoNR}}$ has rate $r = 1$, i.e. f_{PoNR} is tight.*
- *If $c < \log q_{\mathbb{G}}$, then $\mathsf{SIM}_{\mathsf{PoNR}}$ has rate $r = 1 + \frac{1}{\lfloor c/2 \rfloor}$.*

We defer the proofs of Lemma 3 and Theorems 5 and 6 to Appendix B. They follow the same approaches as the correspondings proofs for f_{Vand}, but they are complicated by probabilistic arguments over the randomness of H_{p} and H_{x}.

Note that for all parameters we consider reasonable (e.g. $\kappa = 128$, $t = 30$, and $\log p = 256$) any $N \geq 1$ satsifies the condition $N \geq \frac{\kappa+2t}{\log(p-1)}$. For a taste of Theorem 6's inequality between N and c, if $\log q_{\mathsf{H}} = 128$ and other parameters are as above, then this condition is approximated by $1.66c \leq N \leq 2c$.

3.3 Committed One-Way Functions

Looking ahead to our TFA-KE application, our tight OWFs will be evaluated on some $(\overline{\mathsf{pw}}, s)$ at user registration time to create the server's secret file $F := f(\mathsf{sid}, \overline{\mathsf{pw}}, s)$. The server will also store an ElGamal commitment to F, its associated decommitment, and a NIZK that the commitment is valid for some $(\overline{\mathsf{pw}}, s)$. Then, in each TFA-KE session, the server will send the client that commitment, that proof, and an additional proof that they are executing correctly relative to the F committed therein.

As analyzed thus far, we have no assurance that f_{Vand} and f_{PoNR} remain tight OWFs against an environment that sees an ElGamal commitment to F before the LeakVal event and later sees the associated decommitment upon LeakVal. As described in Figure 3, $\mathsf{SIM}_{\mathsf{Vand}}$ does not simulate these values. We therefore define compiler $\mathsf{CommittedOwf}[f, \mathsf{COM}]$ that, given function f and commitment scheme COM, compiles a construction matching the syntax of $\mathcal{F}_{\mathsf{OWF}}$.

[13] When $N = 1$, f_{PoNR} resembles the Naor-Reingold PRF [39]. We therefore name our function "product of Naor-Reingold", or PoNR.

Definition 5. $\mathsf{CommittedOwf}[f, \mathsf{COM}]$ *is the following protocol.*

- *On* $(\textsc{Eval}, \mathsf{sid}, x)$, P *does: compute* $F := f(\mathsf{sid}, x)$, $(C_F, d_F) \leftarrow \mathsf{COM}(F)$, *store* $(\mathsf{sid}, F, C_F, d_F)$, *and send* C_F *to* $\mathcal{A}^*$.
- *On* $(\textsc{Test}, \mathsf{sid}, x^*)$, P *does: retrieve* $(\mathsf{sid}, F, C_F, d_F)$ *and compute* $F^* := f(\mathsf{sid}, x^*)$; *if* $F^* = F$, *return "correct"; otherwise, return "wrong".*

Lemma 4. *The claims of Theorems 1 and 2 are true for* $\mathsf{CommittedOwf}[f_{\mathsf{Vand}}, \mathsf{ElG}]$ *in addition to* f_{Vand}. *The claims of Theorems 5 and 6 are true for* $\mathsf{CommittedOwf}[f_{\mathsf{PoNR}}, \mathsf{ElG}]$ *in addition to* f_{PoNR}.

We defer the proof of Lemma 4 to Appendix C. In brief, this proof uses the hiding property of ElGamal commitments to show that, in the GGM, all algorithms have negligible probability of using the $\mathbb{G}$ elements of these commitments in a way that disrupts the simulation strategy of $\mathsf{SIM}_{\mathsf{Vand}}$.

4 The TFA-KE Functionality

We define TFA-KE in the setting of Universal Composability (UC) via an ideal functionality $\mathcal{F}_{\mathsf{TFA\text{-}KE}}$ shown in Figure 4. A real-world TFA-KE scheme is a tuple of efficient algorithms $(\mathsf{Init}, \mathsf{AuthCodeGen}, \mathsf{Auth}_{\mathsf{C}}, \mathsf{Auth}_{\mathsf{S}})$, the last two interactive, which respectively correspond to password file initialization by server S, generation of a one-time authentication code by device D, and client C and server S on-line authentication. In the UC formalization, these algorithms implement respectively commands StoreFile, GetCode, CSession, and SSession, which the environment issues to, respectively, parties S, D, C, and S.[14]

The starting point for $\mathcal{F}_{\mathsf{TFA\text{-}KE}}$ is the well-established $\mathcal{F}_{\mathsf{aPAKE}}$ UC functionality of [25], which we now briefly recall. Distinct instances of aPAKE are distinguished by session identifiers sid, which all parties use as an input. In practice it is typical for servers to natively have unique identifiers, e.g. domain names, and for servers to distinguish users by unique identifiers, e.g. usernames or emails. Instance identifer sid could be the concatenation of these two identifiers, but if that doesn't guarantee uniqueness then sid can include a random nonce which the server stores and the client must retrieve to run the TFA-KE protocol.

$\mathcal{F}_{\mathsf{aPAKE}}$ extends UC PAKE to the asymmetric setting, where the server stores a password file (e.g. a password hash) rather than a plaintext password. Compromise of this file is modeled by CompromiseFile. That interface is observable by the environment, which ensures that it can only be used by the ideal adversary at the time of server compromise by the real-world adversary. An offline attack against a compromised file is modeled by the OfflineTest interface, by which an adversary who has compromised the server can test a single pw guess against the stolen file. Honest clients and servers perform online sessions via the CSession and SSession commands. An online attack against an honest session is modeled by the TestPwd interface, to which the adversary must

[14] We further discuss TFA-KE syntax and intuitive security properties in Appendix D.

The functionality internally keeps a random function $\mathsf{R}^{\mathsf{sid}} : \mathbb{N} \to \{0,1\}^t$ for every sid.

File Storage

- On $(\textsc{StoreFile}, \mathsf{sid}, \overline{\mathsf{pw}})$ from S, if this is the first StoreFile message, record $\langle \textsc{file}, \mathsf{sid}, \mathsf{S}, \overline{\mathsf{pw}} \rangle$, mark it uncompromised, and send $(\textsc{StoreFile}, \mathsf{sid}, \mathsf{S})$ to $\mathcal{A}^*$.

Server Leakage

- On $(\textsc{CompromiseFile}, \mathsf{sid})$ from $\mathcal{A}^*$, if there is no record $\langle \textsc{file}, \mathsf{sid}, \mathsf{S}, \overline{\mathsf{pw}} \rangle$, return "no password file" to $\mathcal{A}^*$. Otherwise, if the record is marked uncompromised, mark it compromised; regardless,
  - If there is a record $(\textsc{offline}, \overline{\mathsf{pw}}, T^*, \mathsf{R}^{\mathsf{sid}}(T^*))$, send $(\overline{\mathsf{pw}}, T^*, \mathsf{R}^{\mathsf{sid}}(T^*))$ to $\mathcal{A}^*$.
  - Otherwise return "password file stolen" to $\mathcal{A}^*$.
- On $(\textsc{EvalCodeDiff}, \mathsf{sid}, T_1, T_2)$ from $\mathcal{A}^*$, if there is a record $\langle \textsc{file}, \mathsf{sid}, \mathsf{S}, \overline{\mathsf{pw}} \rangle$ marked compromised, send $\Delta = (\mathsf{R}^{\mathsf{sid}}(T_2) - \mathsf{R}^{\mathsf{sid}}(T_1))$ to $\mathcal{A}^*$.
- On $(\textsc{OfflineTest}, \mathsf{sid}, \mathsf{pw}^*, T^*, \mathsf{code}^*)$ from $\mathcal{A}^*$, do:
  - If there is a record $\langle \textsc{file}, \mathsf{sid}, \mathsf{S}, \overline{\mathsf{pw}} \rangle$ marked compromised, do: if $\mathsf{pw}^* = \overline{\mathsf{pw}}$ and $\mathsf{code}^* = \mathsf{R}^{\mathsf{sid}}(T^*)$, return "correct guess" to $\mathcal{A}^*$; otherwise return "wrong guess."
  - Otherwise record $(\textsc{offline}, \mathsf{pw}^*, T^*, \mathsf{code}^*)$.

Password Authentication

- On $(\textsc{GetCode}, \mathsf{sid}, T)$ from D, send $\mathsf{R}^{\mathsf{sid}}(T)$ to D.
- On $(\textsc{CSession}, \mathsf{sid}, \mathsf{ssid}, \mathsf{S}, \mathsf{pw}, \mathsf{code})$ from C, send $(\textsc{CSession}, \mathsf{sid}, \mathsf{ssid}, \mathsf{C}, \mathsf{S})$ to $\mathcal{A}^*$. If this is the first CSession query for ssid, record $\langle \mathsf{ssid}, \mathsf{C}, \mathsf{S}, \mathsf{pw}, \mathsf{code} \rangle$ and mark it fresh.
- On $(\textsc{SSession}, \mathsf{sid}, \mathsf{ssid}, \mathsf{C}, T)$ from S, retrieve $\langle \textsc{file}, \mathsf{sid}, \mathsf{S}, \overline{\mathsf{pw}} \rangle$, and send $(\textsc{SSession}, \mathsf{sid}, \mathsf{ssid}, \mathsf{S}, \mathsf{C})$ to $\mathcal{A}^*$. If this is the first SSession query for ssid, record $\langle \mathsf{ssid}, \mathsf{S}, \mathsf{C}, \overline{\mathsf{pw}}, \mathsf{R}^{\mathsf{sid}}(T) \rangle$ and mark it fresh.

Active Session Attacks

- On $(\textsc{TestPwd}, \mathsf{sid}, \mathsf{ssid}, \mathsf{P}, \mathsf{pw}^*, \mathsf{code}^*)$ from $\mathcal{A}^*$, if there is a record $\langle \mathsf{ssid}, \mathsf{P}, \mathsf{P}', \mathsf{pw}, \mathsf{code} \rangle$ marked fresh, do: if $\mathsf{pw}^* = \mathsf{pw}$ and $\mathsf{code}^* = \mathsf{code}$, mark it compromised and return "correct guess" to $\mathcal{A}^*$; otherwise mark it interrupted and return "wrong guess" to $\mathcal{A}^*$.
- On $(\textsc{Impersonate}, \mathsf{sid}, \mathsf{ssid}, T, \Delta)$ from $\mathcal{A}^*$, if there is a record $\langle \mathsf{ssid}, \mathsf{C}, \mathsf{S}, \mathsf{pw}, \mathsf{code} \rangle$ marked fresh, do: if there is a record $\langle \textsc{file}, \mathsf{sid}, \mathsf{S}, \overline{\mathsf{pw}} \rangle$ marked compromised, $\mathsf{pw} = \overline{\mathsf{pw}}$ and $\mathsf{code} = \mathsf{R}^{\mathsf{sid}}(T) + \Delta$, mark $\langle \mathsf{ssid}, \mathsf{C}, \mathsf{S}, \mathsf{pw}, \mathsf{code} \rangle$ compromised and return "correct guess" to $\mathcal{A}^*$; otherwise mark it interrupted and return "wrong guess" to $\mathcal{A}^*$.

Key Generation and Authentication

- On $(\textsc{NewKey}, \mathsf{sid}, \mathsf{ssid}, \mathsf{P}, \mathsf{SK}^*)$ from $\mathcal{A}^*$, if there is a record $\langle \mathsf{ssid}, \mathsf{P}, \mathsf{P}', \mathsf{pw}, \mathsf{code} \rangle$ not marked completed, do:
  - If the record is marked compromised set $\mathsf{SK} := \mathsf{SK}^*$.
  - If the record is marked fresh, a $(\mathsf{sid}, \mathsf{ssid}, \mathsf{SK}')$ tuple was sent to P', and at that time there was a record $\langle \mathsf{ssid}, \mathsf{P}', \mathsf{P}, \mathsf{pw}, \mathsf{code} \rangle$ marked fresh, set $\mathsf{SK} := \mathsf{SK}'$.
  - Otherwise choose $\mathsf{SK} \leftarrow_{\$} \{0,1\}^{\ell}$.

  Finally, mark $\langle \mathsf{ssid}, \mathsf{P}, \mathsf{P}', \mathsf{pw}, \mathsf{code} \rangle$ completed and send $(\mathsf{sid}, \mathsf{ssid}, \mathsf{SK})$ to P.

Fig. 4. Functionality $\mathcal{F}_{\mathsf{TFA\text{-}KE}}$, with additions relative to $\mathcal{F}_{\mathsf{aPAKE}}$ in shadowed text

supply a single pw guess. After server compromise, the adversary can also use the Impersonate interface, which models an online attacker's ability to act like an honest server by using a stolen file. For both TestPwd and Impersonate, a failed online attack results in the attacked session being marked interrupted, which guarantees that only a single pw guess per attacked session is possible.

$\mathcal{F}_{\mathsf{TFA\text{-}KE}}$ augments $\mathcal{F}_{\mathsf{aPAKE}}$ to the two-factor setting, wherein there is a new party D that generates a random t-bit code for every epoch $T \in \mathbb{N}$, using the functionality's internal random function $\mathsf{R}^{\mathsf{sid}} : \mathbb{N} \to \{0,1\}^t$. Most of the changes from $\mathcal{F}_{\mathsf{aPAKE}}$ to $\mathcal{F}_{\mathsf{TFA\text{-}KE}}$ follow naturally (see the shadowed text in Fig. 4):

1. Device D, on message $(\textsc{GetCode}, \mathsf{sid}, T)$, outputs $\mathsf{code} = \mathsf{R}^{\mathsf{sid}}(T)$.
2. Client C supplies a code to CSession, which is included in the session record.
3. Server S supplies an epoch T to SSession, which creates a session record with $\mathsf{code} = \mathsf{R}^{\mathsf{sid}}(T)$, same as output by D for epoch T.
4. In NewKey, a client session and server session output the same key SK only if they agree on both pw *and* code.

Notice that, while servers always run on a correct $\mathsf{code} = \mathsf{R}^{\mathsf{sid}}(T)$, clients are "fallible" in a way that is controlled by the environment, which doesn't have to correctly pass the code generated by D to the client. Note that the UC (a)PAKE notion likewise lets the environment control password inputs, e.g. causing arbitrary password mistyping. Since our model makes code output by D visible to the environment, any realization of $\mathcal{F}_{\mathsf{TFA\text{-}KE}}$ must be secure against environments that may leak some code to the adversary, e.g. due to a lunchtime attack against D. This modeling choice also parallels the treatment of passwords in UC (a)PAKE, which can likewise be leaked by the environment to the adversary.

Online security in $\mathcal{F}_{\mathsf{TFA\text{-}KE}}$ is strengthened compared to $\mathcal{F}_{\mathsf{aPAKE}}$. In particular, an online TestPwd attack requires a correct guess of code in addition to pw. If the adversary doesn't learn $\mathsf{R}^{\mathsf{sid}}(T)$, this reduces the probability of success by a factor of 2^t. Moreover, since $\mathsf{R}^{\mathsf{sid}}$ is a random function, any leaked $\mathsf{R}^{\mathsf{sid}}(T)$, e.g. via a lunchtime attack, can aid online attacks only during epoch T.

If the attacker compromises the server's file then it can impersonate the server. This is modeled by the Impersonate query, and as in the case of an honest server, the attacker must set an epoch T in such an execution. However, we add a relaxation that lets the attacker specify an arbitrary *offset* Δ, and the impersonation succeeds if the client runs on $(\mathsf{pw}, \mathsf{code})$ s.t. $\mathsf{code} = \mathsf{R}^{\mathsf{sid}}(T) + \Delta$ and pw matches password $\overline{\mathsf{pw}}$ encoded in the compromised file.[15] Our TFA-KE constructions need this relaxation, because they indeed allow the adversary to use the compromised file with the code-related information 'shifted' by an arbitrary offset. However, note that regardless of this offset the attacker's inputs still define a *single* client's input pair $\mathsf{pw}, \mathsf{code}$ for which the attack succeeds.

[15] Our notation treats the code space $\{0,1\}^t$ as an abelian group, with group operations denoted $+$ and $-$. Looking ahead, our first TFA-KE instantiation (Section 5.3) casts $\{0,1\}^t$ elements as integers in $\mathbb{Z}_{2^t}$, where $a+b$ and $a-b$ stand for addition and subtraction modulo 2^t, and our second instantiation (Section 5.4) casts $\{0,1\}^t$ elements as bitstrings, where both $a+b$ and $a-b$ stand for bitwise XOR.

Offline security of $\mathcal{F}_{\mathsf{TFA\text{-}KE}}$ is also strengthened compared to $\mathcal{F}_{\mathsf{aPAKE}}$. Inevitably, an adversary who has compromised the server's file can locally execute an authentication protocol to test arbitrary inputs, hence we include (T^*, code^*) in addition to pw^* in the input to OfflineTestPwd. A test is successful only if $\mathsf{pw}^* = \overline{\mathsf{pw}}$ for $\overline{\mathsf{pw}}$ encoded in the compromised file, *and* if $\mathsf{code}^* = \mathsf{R}^{\mathsf{sid}}(T^*)$. If the attacker has not learned $\mathsf{R}^{\mathsf{sid}}(T)$ for any T, their offline dictionary attack is slowed down by a factor of 2^t. As in $\mathcal{F}_{\mathsf{aPAKE}}$, offline attacks can be precomputed, in which case their success or failure is learned upon CompromiseFile.

Finally, we make another relaxation in the form of the EvalCodeDiff interface, by which an attacker who compromised the server file can learn differences between $\mathsf{R}^{\mathsf{sid}}$ values for any two epochs. Notice that EvalCodeDiff does not weaken offline security, since such an attacker is free to perform OfflineTestPwd queries for arbitrary epochs. Interface EvalCodeDiff impacts online security, because if the attacker compromises the server file and learns $\mathsf{R}^{\mathsf{sid}}(T)$ for any single epoch T, then it can supply the correct code $\mathsf{R}^{\mathsf{sid}}(T')$ into *online* authentication attempts for any future epoch $T' \neq T$. However, this doesn't seem like much of a relaxation since the same attacker can perform arbitrary *offline* password queries without using this EvalCodeDiff interface.

5 TFA-KE Protocols

In this section we show our TFA-KE protocols. We begin in Section 5.1 by defining $\mathcal{F}_{\mathsf{CKEM}}$, a flexible building block that can be parameterized for different functions. Then, in Section 5.2, we present $\mathsf{TFA}_{\mathsf{gen}}$, a generic construction of a TFA-KE scheme from any tight OWF given a corresponding realization of $\mathcal{F}_{\mathsf{CKEM}}$. Finally, Sections 5.3 and 5.4 contain the efficient realizations of $\mathcal{F}_{\mathsf{CKEM}}$ needed to instantiate $\mathsf{TFA}_{\mathsf{gen}}$ for our two tight OWF proposals f_{Vand} and f_{PoNR}.

5.1 Conditional KEM

Figure 5 is $\mathcal{F}_{\mathsf{CKEM}}$, an ideal functionality for a Conditional Key Encapsulation Mechanism (CKEM). This functionality is parameterized by a circuit $\mathcal{C} : X \times Y \to \{0,1\}$, which defines the condition that must be satisfied for key transfer to succeed. In anticipation of our TFA-KE application, we label our CKEM parties as server S, who runs on input $y \in Y$, and client C, who runs on input $x \in X$. (In KEM terminology these parties are respectively called a *sender* and a *receiver*.) When S and C interact, S first outputs a random key $\mathsf{SK} \leftarrow_{\$} \{0,1\}^{\kappa}$, and C then outputs key SK'. If $\mathcal{C}(x,y) = 1$, then $\mathsf{SK}' := \mathsf{SK}$; otherwise, $\mathsf{SK}' \leftarrow_{\$} \{0,1\}^{\kappa}$.

$\mathcal{F}_{\mathsf{CKEM}}$ is designed to be realizable over insecure channels, so network adversary $\mathcal{A}^*$ has full control over the communication of S and C and can assume the role of either party in interaction with the other. We relax $\mathcal{F}_{\mathsf{CKEM}}$ with parameter $\mathcal{C}_{\mathsf{Adv}} : X^* \times Y \to \{0,1\}$, which is a "downgraded" condition circuit. $\mathcal{A}^*$, when acting as a dishonest client, learns SK if they supply $x^* \in X^*$ s.t. $\mathcal{C}_{\mathsf{Adv}}(x^*, y) = 1$. $\mathcal{F}_{\mathsf{CKEM}}$ is also parameterized by leakage functions $\mathcal{L} = (\mathcal{L}_{\mathsf{C}}, \mathcal{L}_{\mathsf{S}})$ s.t. $\mathcal{L}_{\mathsf{C}}(x)$ and $\mathcal{L}_{\mathsf{S}}(y)$ are allowed to be leaked by a CKEM protocol to the network adversary.

Functionality $\mathcal{F}_{\mathsf{CKEM}}[\mathcal{C}, \mathcal{C}_{\mathsf{Adv}}, \mathcal{L}]$ is parameterized by a circuit $\mathcal{C} : X \times Y \to \{0,1\}$ for some X, Y, an accompanying 'downgraded' circuit $\mathcal{C}_{\mathsf{Adv}} : X' \times Y \to \{0,1\}$ for some X', and leakage functions $\mathcal{L} = (\mathcal{L}_{\mathsf{C}}, \mathcal{L}_{\mathsf{S}})$. Functionality interacts with honest parties (denoted C and S depending on the protocol role they play) and an adversary $\mathcal{A}^*$.

The functionality is also parameterized by *session identifier*, sid, which we omit from the syntax below for visual clarity.

- On $(\textsc{Seval}, \mathsf{ssid}, y)$ from honest party S for $y \in Y$: If there is no prior record $(\textsc{Seval}, \mathsf{ssid}, ...)$ then save $(\textsc{Seval}, \mathsf{ssid}, \mathsf{S}, y)$ marked fresh and send $(\textsc{Seval}, \mathsf{ssid}, \mathsf{S}, \mathcal{L}_{\mathsf{S}}(y))$ to $\mathcal{A}^*$.
- On $(\textsc{Ceval}, \mathsf{ssid}, x)$ from honest party C for $x \in X$: If there is no prior record $(\textsc{Ceval}, \mathsf{ssid}, ...)$ then save $(\textsc{Ceval}, \mathsf{ssid}, \mathsf{C}, x)$ marked fresh and send $(\textsc{Ceval}, \mathsf{ssid}, \mathsf{C}, \mathcal{L}_{\mathsf{C}}(x))$ to $\mathcal{A}^*$.
- On $(\textsc{Sout}, \mathsf{ssid})$ from $\mathcal{A}^*$: If there are records $(\textsc{Ceval}, \mathsf{ssid}, \mathsf{C}, x)$ and $(\textsc{Seval}, \mathsf{ssid}, \mathsf{S}, y)$ for some $\mathsf{C}, \mathsf{S}, x, y$, both marked fresh, then pick $\mathsf{SK} \leftarrow_{\$} \{0,1\}^{\kappa}$, mark the second record $\mathsf{completed}(\mathsf{SK})$, and output $(\textsc{Sout}, \mathsf{ssid}, \mathsf{SK})$ to S.
- On $(\textsc{Cout}, \mathsf{ssid})$ from $\mathcal{A}^*$: If there are records $(\textsc{Ceval}, \mathsf{ssid}, \mathsf{C}, x)$ marked fresh and $(\textsc{Seval}, \mathsf{ssid}, \mathsf{S}, y)$ marked $\mathsf{completed}(\mathsf{SK})$ for some $\mathsf{C}, \mathsf{S}, x, y, \mathsf{SK}$, then mark the first record $\mathsf{completed}$ and if $\mathcal{C}(x, y) = 1$ then output $(\textsc{Cout}, \mathsf{ssid}, \mathsf{SK})$ to C, else pick $\mathsf{SK}' \leftarrow \{0,1\}^{\kappa}$ and output $(\textsc{Cout}, \mathsf{ssid}, \mathsf{SK}')$ to C.
- On $(\textsc{AdvSeval}, \mathsf{ssid}, y^*, \mathsf{SK}^*)$ from $\mathcal{A}^*$ for $y^* \in Y \cup \{\perp\}$ and $\mathsf{SK}^* \in \{0,1\}^{\kappa}$: If there is a record $(\textsc{Ceval}, \mathsf{ssid}, \mathsf{C}, x)$ marked fresh for some C, x, then mark this record used and if $y^* \neq \perp$ and $\mathcal{C}(x, y^*) = 1$ then output $(\textsc{Cout}, \mathsf{ssid}, \mathsf{SK}^*)$ to C, else pick $\mathsf{SK}' \leftarrow \{0,1\}^{\kappa}$ and output $(\textsc{Cout}, \mathsf{ssid}, \mathsf{SK}')$ to C.
- On $(\textsc{AdvCeval}, \mathsf{ssid}, x^*)$ from $\mathcal{A}^*$ for $x^* \in X' \cup \{\perp\}$: If there is a record $(\textsc{Seval}, \mathsf{ssid}, \mathsf{S}, y)$ marked fresh for some S, y, then pick $\mathsf{SK} \leftarrow_{\$} \{0,1\}^{\kappa}$, mark this record used, output $(\textsc{Sout}, \mathsf{ssid}, \mathsf{SK})$ to S, and if $x^* \neq \perp$ and $\mathcal{C}_{\mathsf{Adv}}(x^*, y) = 1$ then output $(\textsc{Cout}, \mathsf{ssid}, \mathsf{SK})$ to $\mathcal{A}^*$ else output $(\textsc{Cout}, \mathsf{ssid}, \perp)$ to $\mathcal{A}^*$.

Fig. 5. $\mathcal{F}_{\mathsf{CKEM}}[\mathcal{C}, \mathcal{C}_{\mathsf{Adv}}, \mathcal{L}]$: Conditional KEM over insecure channels

5.2 Generic TFA-KE using Conditional KEM

Figure 6 is $\mathsf{TFA}_{\mathsf{gen}}$, a generic TFA-KE protocol from any tight one-way function f. More precisely, we rely on a commitment scheme COM and a compiler $\mathsf{CommittedOwf}$, see Section 3.3, which transforms f into a function that in addition to computing $F = f(x)$ outputs a commitment C_F to F. We require that this "committment-amended function" also realizes the tight OWF model $\mathcal{F}_{\mathsf{OWF}}$. For concreteness we will assume that $\mathsf{CommittedOwf}[f, \mathsf{COM}]$ realizes $\mathcal{F}_{\mathsf{OWF}}$ in the *generic group model* (GGM), as indeed is the case for our $\mathcal{F}_{\mathsf{OWF}}$ proposals in Section 3. Moreover, we require that OWF f can be decomposed as $f = f^* \circ f_{\mathsf{pre}}^{\mathcal{HP}}$ (Definition 3) such that f^* is a collision-resistant function. We specify this decomposition so that $\mathsf{TFA}_{\mathsf{gen}}$ can use NIZKs for knowledge of f^* preimage (i.e. hpw, s) instead of f preimage (i.e. pw, s). Thus we avoid proving knowledge of hash function H_{p} preimages, which would be highly inefficient. Additionally, $\mathsf{TFA}_{\mathsf{gen}}$ relies on random oracles, NIZKs, and a realization of CKEM functionality $\mathcal{F}_{\mathsf{CKEM}}$ parametrized by a circuit $\mathsf{SSEM}[f^*, \mathsf{COM}]$ described below.

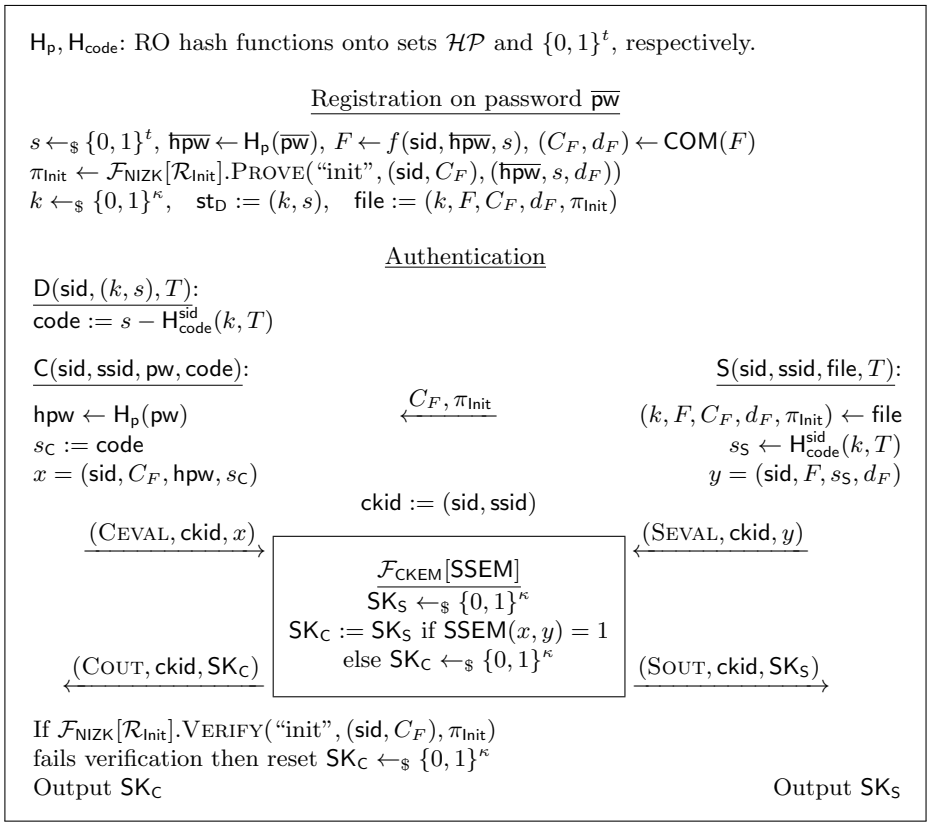


Fig. 6. $\mathsf{TFA_{gen}}$: TFA-KE for a “commitment-amended” OWF function $f = f^* \circ f^{\mathcal{HP}}_{\mathsf{pre}}$

In the initialization, $\mathsf{Init}(1^\kappa, t, \mathsf{sid}, \overline{\mathsf{pw}})$ samples a key $k \leftarrow \{0,1\}^\kappa$ and a t-bit *initialization entropy* value s, evaluates the tight one-way function f on the hashed password and the initialization entropy, i.e. computes $F = f^*(\mathsf{sid}, \overline{\mathsf{hpw}}, s)$ where $\overline{\mathsf{hpw}} = \mathsf{H_p}(\overline{\mathsf{pw}})$ and $\mathsf{H_p}$ is an RO hash onto space $\mathcal{HP}$, commits to F as $(C_F, d_F) \leftarrow \mathsf{COM}(F)$, and sets the device state as $\mathsf{st_D} = (k, s)$ and the server password file as $\mathsf{file} = (k, F, C_F, d_F, \pi_{\mathsf{Init}})$ where π_{Init} is a NIZK for relation $\mathcal{R}_{\mathsf{Init}}$ defined as follows:

$$\mathcal{R}_{\mathsf{Init}} = \{\ (\mathbb{x}, \mathbb{w}) = ((\mathsf{sid}, C_F), ((\overline{\mathsf{hpw}}, s, d_F)) \text{ s.t.} \\ C_F = \mathsf{COM}(F; d_F) \text{ for } F = f^*(\mathsf{sid}, \overline{\mathsf{hpw}}, s)\ \}$$

Given the setup above, the code generation $\mathsf{AuthCodeGen}((k,s), T)$ is an “s-shifted PRF”, $\mathsf{R}^{\mathsf{sid}}_{(k,s)}(T) = s - \mathsf{H}^{\mathsf{sid}}_{\mathsf{code}}(k, T)$ where $\mathsf{H}^{\mathsf{sid}}_{\mathsf{code}}$ is an RO hash onto $\{0,1\}^t$.[16] The authentication protocol (we omit sub-session identifiers from this overview), executed between the client C on $(\mathsf{pw}, \mathsf{code})$ and the server S on

[16] As explained in Section 4, we treat $\{0,1\}^t$ as an abelian group, which can be instantiated with either modular addition or bitwise XOR as the group operation.

(file, T) proceeds as follows. C sets $\mathsf{hpw} := \mathsf{H}_{\mathsf{p}}(\mathsf{pw})$ and $s_{\mathsf{C}} := \mathsf{code}$, and S parses $\mathsf{file} = (k, F, C_F, d_F, \pi_{\mathsf{Init}})$ and sets $s_{\mathsf{S}} := \mathsf{H}^{\mathsf{sid}}_{\mathsf{code}}(k, T)$. Note that if C uses a device code for epoch T then $s_{\mathsf{C}} = s - \mathsf{H}^{\mathsf{sid}}_{\mathsf{code}}(k, T)$ and s_{S} form a *secret-sharing* of the initialization entropy s, namely $s = s_{\mathsf{C}} + s_{\mathsf{S}}$. Server S then sends C_F together with NIZK π_{Init} to C, and the two parties run a CKEM protocol on respective inputs $x = (\mathsf{sid}, C_F, \mathsf{hpw}, s_{\mathsf{C}})$ and $y = (\mathsf{sid}, F, s_{\mathsf{S}}, d_F)$, parametrized by circuit $\mathcal{C} = \mathsf{SSEM}[f^*, \mathsf{COM}]$ and leakage functions $\mathcal{L}_{\mathsf{SSEM}} = (\mathcal{L}_{\mathsf{C}}, \mathcal{L}_{\mathsf{S}})$ defined as follows:

$$\begin{gathered}\mathsf{SSEM}[f^*, \mathsf{COM}](\,(\mathsf{sid}_{\mathsf{C}}, C_F, \mathsf{hpw}, s_{\mathsf{C}})\,,\,(\mathsf{sid}_{\mathsf{S}}, F, s_{\mathsf{S}}, d_F)\,) \;= \\ (\;\mathsf{sid}_{\mathsf{C}} \equiv \mathsf{sid}_{\mathsf{S}} \;\;\wedge\;\; F \equiv f^*(\mathsf{sid}_{\mathsf{C}}, \mathsf{hpw}, s_{\mathsf{C}} + s_{\mathsf{S}}) \;\;\wedge\;\; C_F \equiv \mathsf{COM}(F; d_F)\;)\end{gathered}$$

$$\begin{gathered}\mathcal{L}_{\mathsf{C}}(\mathsf{sid}_{\mathsf{C}}, C_F, \mathsf{hpw}, s_{\mathsf{C}}) = (\mathsf{sid}_{\mathsf{C}}, C_F) \\ \mathcal{L}_{\mathsf{S}}(\mathsf{sid}_{\mathsf{S}}, F, s_{\mathsf{S}}, d_F) = (\mathsf{sid}_{\mathsf{S}}, \mathsf{COM}(F; d_F))\end{gathered}$$

We call the above circuit SSEM for *Secret-Shared Evaluation Match*. Recall, see Section 5.1, that a CKEM functionality parametrized by condition $\mathcal{C}$ that takes inputs x and y from resp. the client C and the server S, outputs a random key SK_{S} to S, and if $\mathcal{C}(x, y) = 1$ then it outputs the same key $\mathsf{SK}_{\mathsf{C}} = \mathsf{SK}_{\mathsf{S}}$ to C, otherwise it samples SK_{C} as an independent random value. Consequently, if $\mathcal{C}$ is defined as above then $\mathsf{SK}_{\mathsf{C}} = \mathsf{SK}_{\mathsf{S}}$ if and only if (1) they use the same sid, (2) C contributes $\mathsf{hpw}, s_{\mathsf{C}}$ s.t. $f^*(\mathsf{sid}, \mathsf{hpw}, s_{\mathsf{C}} + s_{\mathsf{S}})$ evaluates to the server-held value $F = f^*(\mathsf{sid}, \overline{\mathsf{hpw}}, s)$, and (3) C contributes C_F which commits to F.

Recall also, see Section 5.1, that our notion of CKEM allows a malicious client to learn the server's output key if a 'downgraded' condition $\mathcal{C}_{\mathsf{Adv}}$ holds on (x^*, y) where x^* is an input of a malicious client. In our case it suffices to define $\mathcal{C}_{\mathsf{Adv}} = \mathsf{SSEM}_{\mathsf{Adv}}[f^*]$ as follows:

$$\begin{gathered}\mathsf{SSEM}_{\mathsf{Adv}}[f^*](\,(\mathsf{sid}_{\mathsf{C}}, \mathsf{hpw}, s_{\mathsf{C}})\,,\,(\mathsf{sid}_{\mathsf{S}}, F, s_{\mathsf{S}}, d_F)\,) \;= \\ (\;\mathsf{sid}_{\mathsf{C}} \equiv \mathsf{sid}_{\mathsf{S}} \;\;\wedge\;\; F \equiv f^*(\mathsf{sid}_{\mathsf{C}}, \mathsf{hpw}, s_{\mathsf{C}} + s_{\mathsf{S}})\;)\end{gathered}$$

Note that the leakage functions $\mathcal{L}_{\mathsf{C}}, \mathcal{L}_{\mathsf{S}}$ imply that the computation might reveal C's inputs $\mathsf{sid}_{\mathsf{C}}$ and C_F and S's inputs $\mathsf{sid}_{\mathsf{S}}$ and $C_F = \mathsf{COM}(F; d_F)$. Since all these values are public this leakage is innocuous in the context of our protocol. Finally, C verifies that π_{Init} is a valid proof for relation $\mathcal{R}_{\mathsf{Init}}$ on statement (sid, C_F), and if it does not hold then C resets its output SK_{C} as a random key.

Theorem 7. *Protocol* $\mathsf{TFA}_{\mathsf{gen}}$ *realizes* $\mathcal{F}_{\mathsf{TFA\text{-}KE}}$ *in the* $(\mathcal{F}_{\mathsf{CKEM}}[\mathsf{SSEM}[f^*, \mathsf{COM}], \mathsf{SSEM}_{\mathsf{Adv}}[f^*]], \mathcal{F}_{\mathsf{NIZK}}[\mathcal{R}_{\mathsf{Init}}])$*-hybrid world and GGM, assuming* $\mathsf{CommittedOwf}[f, \mathsf{COM}]$ *tightly realizes* $\mathcal{F}_{\mathsf{OWF}}$ *in GGM,* $f = f^* \circ f^{\mathcal{HP}}_{\mathsf{pre}}$ *(Definition 3),* f^* *is collision-resistant,* COM *is a hiding and binding commitment scheme, hash function* $\mathsf{H}_{\mathsf{code}}$ *is modeled as a random oracle, hash function* H_{p} *is modeled as a non-programmable random oracle, and at most* $q_{\mathsf{H}} \leq 2^{(\log|\mathcal{HP}|-\kappa)/2}$ *queries are made to* H_{p}.

We defer the proof of Theorem 7 to Appendix E.

5.3 TFA-KE from f_{Vand}

To instantiate the generic $\mathsf{TFA}_{\mathsf{gen}}$ construction for OWF f_{Vand} (Section 3.1) and commitment scheme EIG, we need a realization of $\mathcal{F}_{\mathsf{CKEM}}[\mathsf{SSEM}[f^*_{\mathsf{Vand}}, \mathsf{EIG}], \mathsf{SSEM}_{\mathsf{Adv}}[f^*_{\mathsf{Vand}}]]$, which we provide as protocol $\mathsf{CKEM}_{\mathsf{Vand}}$ in Figure 7. This protocol is based on the following application of the binomial theorem:

$$\prod_{i=1}^{N} H_i{}^{(a+b)^i} = \prod_{i=1}^{N} H_i{}^{\sum_{j=0}^{i}\binom{i}{j}a^{i-j}b^j} = \prod_{i=1}^{N}\prod_{j=0}^{i} H_i{}^{\binom{i}{j}a^{i-j}b^j}$$
$$= \prod_{j=0}^{N}\left(\prod_{i=\max\{j,1\}}^{N} H_i{}^{\binom{i}{j}a^{i-j}}\right)^{b^j} \tag{5}$$

$\underline{\mathsf{C}(\mathsf{ssid}, (\mathsf{sid}_\mathsf{C}, C_F, \mathsf{hpw}, s_\mathsf{C}))}$: $\qquad\qquad$ $\underline{\mathsf{S}(\mathsf{ssid}, (\mathsf{sid}_\mathsf{S}, F, s_\mathsf{S}, d_F))}$:

$a := \mathsf{hpw}|0|s_\mathsf{C}, \quad (\mathsf{sk}_\mathsf{C}, \mathsf{pk}_\mathsf{C}) \leftarrow \mathsf{EIG.KG}$

$\forall_{j\in\{0,1,\ldots,N\}}$: $g_j := \prod_{i=\max\{j,1\}}^{N} \mathsf{H}^{\mathsf{sid}_\mathsf{C}}_{\mathbb{G}}(i)^{\binom{i}{j}a^{i-j}}$,
$d_{G_j} \leftarrow_\$ \mathbb{Z}_p, \ G_j \leftarrow \mathsf{EIG.Enc}_{\mathsf{pk}_\mathsf{C}}(g_j; d_{G_j})$

$\boldsymbol{d}_G \leftarrow (d_{G_j})_{j\in\{0,1,\ldots,N\}}, \ \boldsymbol{G} \leftarrow (G_j)_{j\in\{0,1,\ldots,N\}}$

$\pi_\mathsf{C} \leftarrow \mathcal{F}_{\mathsf{NIZK}}[\mathcal{R}_\mathsf{C}].\textsc{Prove}(\mathsf{ssid}, (\mathsf{sid}_\mathsf{C}, \mathsf{pk}_\mathsf{C}, \boldsymbol{G}), (\mathsf{hpw}, s_\mathsf{C}, \boldsymbol{d}_G))$

$$\xrightarrow{\mathsf{sid}_\mathsf{C}, \mathsf{pk}_\mathsf{C}, \boldsymbol{G}, \pi_\mathsf{C}}$$

if $\mathsf{sid}_\mathsf{C} \neq \mathsf{sid}_\mathsf{S} \vee \mathcal{F}_{\mathsf{NIZK}}[\mathcal{R}_\mathsf{C}].\textsc{Verify}(\mathsf{ssid}, (\mathsf{sid}_\mathsf{S}, \mathsf{pk}_\mathsf{C}, \boldsymbol{G}), \pi_\mathsf{C}) = 0$
then $\mathsf{SK} \leftarrow_\$ \{0,1\}^\kappa$, **output** SK, and **halt**

$K \leftarrow_\$ \mathbb{G}, \ d_K := \mathsf{H}^{\mathsf{sid}_\mathsf{S}}_{\mathsf{d}}(K), \ C_K \leftarrow \mathsf{EIG.Commit}(K; d_K)$
output $\mathsf{SK} := \mathsf{H}^{\mathsf{sid}_\mathsf{S}}_{\mathsf{SK}}(\mathsf{ssid}, K)$
$C_F \leftarrow \mathsf{EIG.Commit}(F; d_F), \ r_0, r_1 \leftarrow_\$ \mathbb{Z}_p \setminus \{0\}$
$d_{V_0} \leftarrow_\$ \mathbb{Z}_p, \ V_0 \leftarrow \mathsf{EIG.Enc}_{\mathsf{pk}_\mathsf{C}}(K; d_{V_0}) \cdot (\prod_{j=0}^{N}(G_j)^{{s_\mathsf{S}}^j}/F)^{r_0}$
$d_{V_1} \leftarrow_\$ \mathbb{Z}_p, \ V_1 \leftarrow \mathsf{EIG.Enc}_{\mathsf{pk}_\mathsf{C}}(K; d_{V_1}) \cdot (\prod_{j=0}^{N}(G_j)^{(s_\mathsf{S}-2^t)^j}/F)^{r_1}$
$\beta \leftarrow_\$ \{0,1\}, \ (V'_0, V'_1) := (V_\beta, V_{1-\beta})$
$\pi_\mathsf{S} \leftarrow \mathcal{F}_{\mathsf{NIZK}}[\mathcal{R}_\mathsf{S}].\textsc{Prove}(\mathsf{ssid}, (C_F, \mathsf{pk}_\mathsf{C}, \boldsymbol{G}, C_K, (V'_0, V'_1)),$
$(F, d_F, s_\mathsf{S}, (r_0, r_1), K, d_K, (d_{V_0}, d_{V_1}), (V_0, V_1)))$

$$\xleftarrow{C_K, (V'_0, V'_1), \pi_\mathsf{S}}$$

if $\mathcal{F}_{\mathsf{NIZK}}[\mathcal{R}_\mathsf{S}].\textsc{Verify}(\mathsf{ssid}, (C_F, \mathsf{pk}_\mathsf{C}, \boldsymbol{G}, C_K, (V'_0, V'_1)), \pi_\mathsf{S}) = 0$
then $\mathsf{SK}' \leftarrow_\$ \{0,1\}^\kappa$, **output** SK', and **halt**

$K'_0 \leftarrow \mathsf{EIG.Dec}_{\mathsf{sk}_\mathsf{C}}(V'_0), \ K'_1 \leftarrow \mathsf{EIG.Dec}_{\mathsf{sk}_\mathsf{C}}(V'_1)$
if $C_K = \mathsf{EIG.Commit}(K'_0; \mathsf{H}^{\mathsf{sid}_\mathsf{C}}_{\mathsf{d}}(K'_0))$ **then output** $\mathsf{SK}' := \mathsf{H}^{\mathsf{sid}_\mathsf{C}}_{\mathsf{SK}}(\mathsf{ssid}, K'_0)$
else if $C_K = \mathsf{EIG.Commit}(K'_1; \mathsf{H}^{\mathsf{sid}_\mathsf{C}}_{\mathsf{d}}(K'_1))$ **then output** $\mathsf{SK}' := \mathsf{H}^{\mathsf{sid}_\mathsf{C}}_{\mathsf{SK}}(\mathsf{ssid}, K'_1)$
else output $\mathsf{SK}' \leftarrow_\$ \{0,1\}^\kappa$

Fig. 7. $\mathsf{CKEM}_{\mathsf{Vand}}$: Conditional KEM protocol for SSEM of function f^*_{Vand}

Setting $H_i = \mathsf{H}^{\mathsf{sid}}_{\mathbb{G}}(i)$, $a = (\mathsf{hpw}|0|s_{\mathsf{C}})$, and $b = s_{\mathsf{S}}$, Equation 5 suggests an efficient way for two parties to securely evaluate $f^*_{\mathsf{Vand}}(\mathsf{sid}, \mathsf{hpw}, s_{\mathsf{C}} + s_{\mathsf{S}})$. The client computes $g_j := \prod_{i=\max\{j,1\}}^{N} H_i^{\binom{i}{j}a^{i-j}}$ for every $j \in \{0, 1, \ldots, N\}$ and sends the server $(\mathsf{pk}_{\mathsf{C}}, \boldsymbol{G}, \pi_{\mathsf{C}})$ s.t. $\boldsymbol{G}$ is a vector of ElGamal encryptions to the g_j's under public key pk_{C}. π_{C} is a NIZK for relation $\mathcal{R}_{\mathsf{C}}$ s.t. a statement $(\mathsf{sid}, \mathsf{pk}_{\mathsf{C}}, \boldsymbol{G})$ with witness $(\mathsf{hpw}, s_{\mathsf{C}}, \boldsymbol{d}_G)$ belongs to $\mathcal{R}_{\mathsf{C}}$ iff $\boldsymbol{G}$ is correctly computed for sid, hpw, s_{C}, pk_{C}, and encryption randomness $\boldsymbol{d}_G$.

Receiving vector $\boldsymbol{G}$, the server uses the homomorphic property of ElGamal encryption to complete the computation of $F' = f^*_{\mathsf{Vand}}(\mathsf{sid}, \mathsf{hpw}, s_{\mathsf{C}} + s_{\mathsf{S}}) = \prod_{j=0}^{N} (g_j)^{b^j}$. This value is computed in a ciphertext under pk_{C}. To realize $\mathcal{F}_{\mathsf{CKEM}}$, the server must send the client a randomly chosen SK conditioned on whether F' equals the server's input F. To achieve this, the server samples $K \leftarrow_{\$} \mathbb{G}$, defines $\mathsf{SK} := \mathsf{H}^{\mathsf{sid}}_{\mathsf{SK}}(K)$, samples $r \leftarrow_{\$} \mathbb{Z}_p$, and computes a ciphertext V for $K' = K \cdot (F'/F)^r$. K' equals K if $F' = F$ and is random otherwise. The server sends V to the client, who decrypts K' and outputs $\mathsf{SK}' := \mathsf{H}^{\mathsf{sid}}_{\mathsf{SK}}(K')$.[17]

Some details remain unexplained. First, we must compare F with $F' = f^*_{\mathsf{Vand}}(\mathsf{sid}, \mathsf{hpw}, s_{\mathsf{C}} + s_{\mathsf{S}} \bmod 2^t)$, but our protocol adds a and b using arithmetic modulo p, not 2^t. To correct for this, the server performs the above procedure twice in parallel, once with $b = s_{\mathsf{S}}$ and once with $b = s_{\mathsf{S}} - 2^t$. To hide all information about s_{S} from the client, the server randomly flips the order of these two responses. The client now computes two possibilities for K' and must somehow decide which one to use. Therefore, the server sends the client a key confirmation $C_K \leftarrow \mathsf{EIG.Commit}(K; \mathsf{H}^{\mathsf{sid}}_{\mathsf{d}}(K))$. The client uses it to check which K' is correct, and outputs the corresponding $\mathsf{SK}' := \mathsf{H}^{\mathsf{sid}}_{\mathsf{SK}}(K')$. If neither K' matches C_K, the client instead outputs a random $\mathsf{SK}' \leftarrow_{\$} \{0,1\}^{\kappa}$. Finally, the server must send a proof of knowledge π_{S} for relation $\mathcal{R}_{\mathsf{S}}$, which proves their correct behavior. In particular, it is essential that the server uses the F committed in C_F and the K committed in C_K to compute both V ciphertexts. If a dishonest server were able to misbehave on one of the two V computations, they could gain information about the relative values of s_{C}, s_{S}, and the s embedded in F by observing the success or failure of the protocol, i.e. whether or not $\mathsf{SK}' = \mathsf{SK}$.

Appendix F contains formal definitions of relations $\mathcal{R}_{\mathsf{C}}$ and $\mathcal{R}_{\mathsf{S}}$ and a discussion of how NIZKs for them are efficiently realized by standard techniques (as is a NIZK for relation $\mathcal{R}_{\mathsf{Init}}$ of $\mathsf{TFA}_{\mathsf{gen}}$ for OWF f_{Vand}).

Theorem 8. *Protocol* $\mathsf{CKEM}_{\mathsf{Vand}}$ *realizes* $\mathcal{F}_{\mathsf{CKEM}}[\mathsf{SSEM}[f^*_{\mathsf{Vand}}, \mathsf{EIG}], \mathsf{SSEM}_{\mathsf{Adv}}[f^*_{\mathsf{Vand}}]]$ *in the* $(\mathcal{F}_{\mathsf{NIZK}}[\mathcal{R}_{\mathsf{C}}], \mathcal{F}_{\mathsf{NIZK}}[\mathcal{R}_{\mathsf{S}}])$*-hybrid world, assuming hash functions* H_{d} *and* H_{SK} *are modeled as random oracles and DDH is hard in* $\mathbb{G}$.

We defer the proof of Theorem 8 to Appendix G.

[17] Note the cruciality of the $j = 0$ factor in the outer product of Equation 5. $0^j = 0$ for all j, except $0^0 = 1$. Thus, without the $j = 0$ factor, setting $b = 0$ would cause $F' = 1$ regardless of the client's input.

5.4 TFA-KE from f_{PoNR}

An instantiation of protocol $\mathsf{TFA}_{\mathsf{gen}}$ for OWF f_{PoNR} (Section 3.2) and commitment scheme EIG requires realization of $\mathcal{F}_{\mathsf{CKEM}}[\mathsf{SSEM}[f^*_{\mathsf{PoNR}}, \mathsf{EIG}], \mathsf{SSEM}_{\mathsf{Adv}}[f^*_{\mathsf{PoNR}}]]$, which we provide as protocol $\mathsf{CKEM}_{\mathsf{PoNR}}$ in Figure 8. This protocol follows the same general structure as $\mathsf{CKEM}_{\mathsf{Vand}}$ (Section 5.3). In particular, the client on input $(\mathsf{hpw}, s_{\mathsf{C}})$ produces some ElGamal ciphertexts which the server on input (F, s_{S}) uses to compute an encryption of $F' = f^*_{\mathsf{PoNR}}(\mathsf{sid}, \mathsf{hpw}, s_{\mathsf{C}} \oplus s_{\mathsf{S}})$. The server then samples $K \leftarrow_{\$} \mathbb{G}$, defines $\mathsf{SK} := \mathsf{H}^{\mathsf{sid}}_{\mathsf{SK}}(K)$, samples $r \leftarrow_{\$} \mathbb{Z}_p$, and computes a ciphertext V for $K' = K \cdot (F'/F)^r$. The server sends V to the client, who decrypts K' and outputs $\mathsf{SK}' := \mathsf{H}^{\mathsf{sid}}_{\mathsf{SK}}(K')$ (as in $\mathsf{CKEM}_{\mathsf{Vand}}$, the client first verifies K' against key confirmation C_K).

$\mathsf{CKEM}_{\mathsf{PoNR}}$ differs from $\mathsf{CKEM}_{\mathsf{Vand}}$ in how F' is computed. Recalling that OWF f_{PoNR} resembles a product of N Naor-Reingold PRFs [39], we use the approach introduced by [22] for obliviously evaluating the Naor-Reingold PRF using oblivious transfer (OT), and we repeat the procedure in parallel for each $i \in [N]$. For each bit $s_{\mathsf{C}}[j]$ of s_{C}, the client picks $\delta_{i,j} \leftarrow_{\$} \mathbb{Z}^*_p$ and acts as an OT sender on messages $(m_0, m_1) = (\delta_{i,j}, \delta_{i,j} \cdot \mathsf{H}_{\mathsf{x}}(i,j))$. If bit $s_{\mathsf{C}}[j] = 1$, then the client reverses the order of the messages. The server, acting as OT receiver, uses $s_{\mathsf{S}}[j]$ as their choice bit. Additionally, the client computes $g_i := \mathsf{H}^{\mathsf{sid}}_{\mathbb{G}}(i)^{\mathsf{hpw}_i / \prod_{j \in [t]} \delta_{i,j}}$ and sends the server an encryption G_i of it. By exponentiating G_i to the product of their received OT messages, the server computes a ciphertext for $\mathsf{H}^{\mathsf{sid}}_{\mathbb{G}}(i)^{\mathsf{hpw}_i \cdot \prod_{j \in [t], s_{\mathsf{C}}[j] \oplus s_{\mathsf{S}}[j] = 1} \mathsf{H}_{\mathsf{x}}(i,j)}$. The server combines these ciphertexts for all $i \in [N]$ (along with an encryption $\hat{G}$ of $\hat{g} := \prod_{i \in [N]} \hat{\mathsf{H}}^{\mathsf{sid}}_{\mathbb{G}}(i)^{\mathsf{hpw}_i}$ additionally sent by the client) to create the encryption of F'.

Like $\mathsf{CKEM}_{\mathsf{Vand}}$, security of $\mathsf{CKEM}_{\mathsf{PoNR}}$ requires both parties to create NIZKs that prove their correct behavior and enable efficient extraction of their inputs. We therefore upgrade the OT building block to committed OT (COT; Appendix A). The client's proof π_{C} is for relation $\mathcal{R}_{\mathsf{C}}$ and proves that $(G_i)_{i \in [N]}$, $\hat{G}$, and all COT inputs are correct for some $\mathsf{hpw}, s_{\mathsf{C}}$. The server's proof π_{S} is for relation $\mathcal{R}_{\mathsf{S}}$ and proves that V is correctly computed for the committed F, the committed K, and the received COT outputs.

Appendix F contains formal definitions of relations $\mathcal{R}_{\mathsf{C}}$ and $\mathcal{R}_{\mathsf{S}}$ and a discussion of how NIZKs for them are efficiently realized by standard techniques (as is a NIZK for relation $\mathcal{R}_{\mathsf{Init}}$ of $\mathsf{TFA}_{\mathsf{gen}}$ for OWF f_{PoNR}).

Theorem 9. *Protocol* $\mathsf{CKEM}_{\mathsf{PoNR}}$ *realizes* $\mathcal{F}_{\mathsf{CKEM}}[\mathsf{SSEM}[f^*_{\mathsf{PoNR}}, \mathsf{EIG}], \mathsf{SSEM}_{\mathsf{Adv}}[f^*_{\mathsf{PoNR}}]]$ *in the* $(\mathcal{F}_{\mathsf{COT}}[\mathsf{Ped}], \mathcal{F}_{\mathsf{NIZK}}[\mathcal{R}_{\mathsf{C}}], \mathcal{F}_{\mathsf{NIZK}}[\mathcal{R}_{\mathsf{S}}])$*-hybrid world, assuming hash functions* H_{d} *and* H_{SK} *are modeled as random oracles and DDH is hard in* $\mathbb{G}$*.*

We defer the proof of Theorem 9 to Appendix H.

References

1. M. Abdalla and D. Pointcheval. Simple password-based encrypted key exchange protocols. In A. Menezes, editor, *CT-RSA 2005*, volume 3376 of *LNCS*, pages 191–208. Springer, Berlin, Heidelberg, Feb. 2005.

$\underline{\mathsf{C}(\mathsf{ssid}, (\mathsf{sid}_\mathsf{C}, C_F, \mathsf{hpw}, s_\mathsf{C}))}$: $\underline{\mathsf{S}(\mathsf{ssid}, (\mathsf{sid}_\mathsf{S}, F, s_\mathsf{S}, d_F))}$:

$(\mathsf{sk}_\mathsf{C}, \mathsf{pk}_\mathsf{C}) \leftarrow \mathsf{EIG.KG}$

$\forall_{i\in[N]}$: $\delta_{i,1}, \ldots, \delta_{i,t} \leftarrow_\$ \mathbb{Z}_p^*, \quad \Delta_i := \prod_{j\in[t]} \delta_{i,j}$

$g_i := \mathsf{H}_\mathbb{G}^{\mathsf{sid}_\mathsf{C}}(i)^{\mathsf{hpw}_i/\Delta_i}$

$d_{G_i} \leftarrow_\$ \mathbb{Z}_p, \ G_i \leftarrow \mathsf{EIG.Enc}_{\mathsf{pk}_\mathsf{C}}(g_i; d_{G_i})$

$\boldsymbol{\delta} \leftarrow (\delta_{i,j})_{i\in[N],j\in[t]}, \ \boldsymbol{d}_G \leftarrow (d_{G_i})_{i\in[N]}, \ \boldsymbol{G} \leftarrow (G_i)_{i\in[N]}$

$\hat{g} := \prod_{i\in[N]} \hat{\mathsf{H}}_\mathbb{G}^{\mathsf{sid}_\mathsf{C}}(i)^{\mathsf{hpw}_i}, \ d_{\hat{G}} \leftarrow_\$ \mathbb{Z}_p, \ \hat{G} := \mathsf{EIG.Enc}_{\mathsf{pk}_\mathsf{C}}(\hat{g}; d_{\hat{G}})$

$\forall_{j\in[t]}$: $\forall_{i\in[N]}$: $(m_{i,j,s_\mathsf{C}[j]}, m_{i,j,1-s_\mathsf{C}[j]}) := (\delta_{i,j}, \delta_{i,j} \cdot \mathsf{H}_\mathsf{x}(i,j))$

$(C_{m_{j,0}}, d_{m_{j,0}}) \leftarrow \mathsf{Ped}^N((m_{i,j,0})_{i\in[N]})$

$(C_{m_{j,1}}, d_{m_{j,1}}) \leftarrow \mathsf{Ped}^N((m_{i,j,1})_{i\in[N]})$

$\boldsymbol{d}_m \leftarrow (d_{m_{j,0}}, d_{m_{j,1}})_{j\in[t]}, \ \boldsymbol{C}_m \leftarrow (C_{m_{j,0}}, C_{m_{j,1}})_{j\in[t]}$

$\pi_\mathsf{C} \leftarrow \mathcal{F}_{\mathsf{NIZK}}[\mathcal{R}_\mathsf{C}].\textsc{Prove}(\mathsf{ssid}, (\mathsf{sid}_\mathsf{C}, \mathsf{pk}_\mathsf{C}, \boldsymbol{G}, \hat{G}, \boldsymbol{C}_m), (\mathsf{hpw}, s_\mathsf{C}, \boldsymbol{d}_G, d_{\hat{G}}, \boldsymbol{\delta}, \boldsymbol{d}_m))$

$\xrightarrow{\mathsf{sid}_\mathsf{C}, \mathsf{pk}_\mathsf{C}, \boldsymbol{G}, \hat{G}, \pi_\mathsf{C}}$

$\forall_{j\in[t]}$: $\xrightarrow{(m_{i,j,0})_{i\in[N]}, d_{m_{j,0}}}$ $(m_{i,j,1})_{i\in[N]}, d_{m_{j,1}}$ | $\mathcal{F}_{\mathsf{COT}}$ | $\xleftarrow{s_\mathsf{S}[j]}$ $\xrightarrow{(m_{i,j})_{i\in[N]}, d_{m_j}, C_{m_{j,1-s_\mathsf{S}[j]}}}$

$\boldsymbol{m}^\mathsf{S} \leftarrow (m_{i,j})_{i\in[N],j\in[t]}, \ \boldsymbol{d}_m^\mathsf{S} \leftarrow (d_{m_j})_{j\in[t]}$

$\forall_{j\in[t]}$: $C_{m_{j,s_\mathsf{S}[j]}} \leftarrow \mathsf{Ped}^N((m_{i,j})_{i\in[N]}; d_{m_j})$

$\boldsymbol{C}_m \leftarrow (C_{m_{j,0}}, C_{m_{j,1}})_{j\in[t]}$

if $\mathsf{sid}_\mathsf{C} \neq \mathsf{sid}_\mathsf{S} \vee \mathcal{F}_{\mathsf{NIZK}}[\mathcal{R}_\mathsf{C}].\textsc{Verify}(\mathsf{ssid}, (\mathsf{sid}_\mathsf{S}, \mathsf{pk}_\mathsf{C}, \boldsymbol{G}, \hat{G}, \boldsymbol{C}_m), \pi_\mathsf{C}) = 0$
then $\mathsf{SK} \leftarrow_\$ \{0,1\}^\kappa$, **output** SK, and **halt**

$K \leftarrow_\$ \mathbb{G}, \ d_K := \mathsf{H}_\mathsf{d}^{\mathsf{sid}_\mathsf{S}}(K), \ C_K \leftarrow \mathsf{EIG.Commit}(K; d_K)$

output $\mathsf{SK} := \mathsf{H}_\mathsf{SK}^{\mathsf{sid}_\mathsf{S}}(\mathsf{ssid}, K)$

$C_F \leftarrow \mathsf{EIG.Commit}(F; d_F), \ r \leftarrow_\$ \mathbb{Z}_p \setminus \{0\}$

$d_V \leftarrow_\$ \mathbb{Z}_p, \ V \leftarrow \mathsf{EIG.Enc}_{\mathsf{pk}_\mathsf{C}}(K; d_V) \cdot \left(\hat{G} \cdot \prod_{i=1}^N (G_i)^{\prod_{j\in[t]} m_{i,j}} / F\right)^r$

$\pi_\mathsf{S} \leftarrow \mathcal{F}_{\mathsf{NIZK}}[\mathcal{R}_\mathsf{S}].\textsc{Prove}(\mathsf{ssid}, (C_F, \mathsf{pk}_\mathsf{C}, \boldsymbol{G}, \hat{G}, \boldsymbol{C}_m, C_K, V),$
$(F, d_F, s_\mathsf{S}, \boldsymbol{m}^\mathsf{S}, \boldsymbol{d}_m^\mathsf{S}, r, K, d_K, d_V))$

$\xleftarrow{C_K, V, \pi_\mathsf{S}}$

if $\mathcal{F}_{\mathsf{NIZK}}[\mathcal{R}_\mathsf{S}].\textsc{Verify}(\mathsf{ssid}, (C_F, \mathsf{pk}_\mathsf{C}, \boldsymbol{G}, \hat{G}, \boldsymbol{C}_m, C_K, V), \pi_\mathsf{S}) = 0$
then $\mathsf{SK}' \leftarrow_\$ \{0,1\}^\kappa$, **output** SK', and **halt**

$K' \leftarrow \mathsf{EIG.Dec}_{\mathsf{sk}_\mathsf{C}}(V)$

if $C_K = \mathsf{EIG.Commit}(K'; \mathsf{H}_\mathsf{d}^{\mathsf{sid}_\mathsf{C}}(K'))$ **then output** $\mathsf{SK}' := \mathsf{H}_\mathsf{SK}^{\mathsf{sid}_\mathsf{C}}(\mathsf{ssid}, K')$

else output $\mathsf{SK}' \leftarrow_\$ \{0,1\}^\kappa$

Fig. 8. $\mathsf{CKEM}_{\mathsf{PoNR}}$: Conditional KEM protocol for SSEM of function f^*_{PoNR}

2. N. Alon, S. Hoory, and N. Linial. The Moore Bound for Irregular Graphs. *Graphs and Combinatorics*, 18(1):53–57, Mar. 2002.
3. T. Attema and R. Cramer. Compressed sigma-protocol theory and practical application to plug & play secure algorithmics. In *Advances in Cryptology – CRYPTO 2020: 40th Annual International Cryptology Conference, CRYPTO 2020, Santa Barbara, CA, USA, August 17–21, 2020, Proceedings, Part III*, page 513–543, Berlin, Heidelberg, 2020. Springer-Verlag.
4. M. Bellare, D. Pointcheval, and P. Rogaway. Authenticated key exchange secure against dictionary attacks. In B. Preneel, editor, *EUROCRYPT 2000*, volume 1807 of *LNCS*, pages 139–155. Springer, Berlin, Heidelberg, May 2000.
5. S. M. Bellovin and M. Merritt. Encrypted key exchange: Password-based protocols secure against dictionary attacks. In *1992 IEEE Symposium on Security and Privacy*, pages 72–84. IEEE Computer Society Press, May 1992.
6. F. Benhamouda and D. Pointcheval. Verifier-based password-authenticated key exchange: New models and constructions. Cryptology ePrint Archive, Report 2013/833, 2013.
7. J. Bonneau. The science of guessing: Analyzing an anonymized corpus of 70 million passwords. In *Proceedings of the 2012 IEEE Symposium on Security and Privacy*, SP '12, page 538–552, USA, 2012. IEEE Computer Society.
8. X. Boyen, S. Jarecki, P. Nazarian, J. Xu, and T. Zheng. Two-factor authentication can harden servers against offline password search. In J. Daemen and E. Thomé, editors, *Advances in Cryptology – EUROCRYPT 2026*, pages 128–158, Cham, 2026. Springer Nature Switzerland.
9. V. Boyko, P. D. MacKenzie, and S. Patel. Provably secure password-authenticated key exchange using Diffie-Hellman. In B. Preneel, editor, *EUROCRYPT 2000*, volume 1807 of *LNCS*, pages 156–171. Springer, Berlin, Heidelberg, May 2000.
10. T. Bradley, S. Jarecki, and J. Xu. Strong asymmetric PAKE based on trapdoor CKEM. In A. Boldyreva and D. Micciancio, editors, *Advances in Cryptology – CRYPTO 2019*, pages 798–825, Cham, 2019. Springer International Publishing.
11. C. Cachin and J. Camenisch. Optimistic Fair Secure Computation. In M. Bellare, editor, *Advances in Cryptology — CRYPTO 2000*, pages 93–111, Berlin, Heidelberg, 2000. Springer.
12. J. Camenisch and M. Stadler. Proof systems for general statements about discrete logarithms. Technical Report No. 260, March 1997. `ftp://ftp.inf.ethz.ch/pub/publications/tech-reports/`.
13. R. Canetti, S. Halevi, J. Katz, Y. Lindell, and P. D. MacKenzie. Universally composable password-based key exchange. In R. Cramer, editor, *EUROCRYPT 2005*, volume 3494 of *LNCS*, pages 404–421. Springer, Berlin, Heidelberg, May 2005.
14. Y.-H. Chen and Y. Lindell. Optimizing and Implementing Fischlin's Transform for UC-Secure Zero Knowledge. *IACR Communications in Cryptology*, 1(2), July 2024.
15. M. Christ, F. Baldimtsi, K. K. Chalkias, D. Maram, A. Roy, and J. Wang. SoK: Zero-knowledge range proofs. Cryptology ePrint Archive, Paper 2024/430, 2024.
16. R. Cramer, I. Damgård, and Y. Ishai. Share conversion, pseudorandom secret-sharing and applications to secure computation. In J. Kilian, editor, *Theory of Cryptography*, pages 342–362, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.
17. C. Crepeau. Verifiable Disclosure of Secrets and Applications. In J.-J. Quisquater and J. Vandewalle, editors, *Advances in Cryptology — EUROCRYPT '89*, pages 150–154, Berlin, Heidelberg, 1989. Springer.

18. B. F. Dos Santos, Y. Gu, S. Jarecki, and H. Krawczyk. Asymmetric PAKE with low computation and communication. In *Advances in Cryptology - EUROCRYPT 2022*, pages 127–156, 2022.
19. P. Erdös and H. Sachs. Reguläre Graphen gegebener Taillenweite mit minimaler Knotenzahl. *Wissenschaftliche Zeitschrift der Martin-Luther-Universität Halle-Wittenberg*, Math.-Nat. XII(3):251–258, Mar. 1963.
20. M. Fischlin. Communication-Efficient Non-interactive Proofs of Knowledge with Online Extractors. In V. Shoup, editor, *Advances in Cryptology – CRYPTO 2005*, pages 152–168, Berlin, Heidelberg, 2005. Springer.
21. W. Ford and B. S. Kaliski, Jr. Server-assisted generation of a strong secret from a password. In *9th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE 2000)*, pages 176–180, Gaithersburg, MD, USA, June 4–16, 2000. IEEE Computer Society.
22. M. J. Freedman, Y. Ishai, B. Pinkas, and O. Reingold. Keyword Search and Oblivious Pseudorandom Functions. In J. Kilian, editor, *Theory of Cryptography*, pages 303–324, Berlin, Heidelberg, 2005. Springer.
23. Z. Füredi and M. Simonovits. The History of Degenerate (Bipartite) Extremal Graph Problems. In L. Lovász, I. Z. Ruzsa, and V. T. Sós, editors, *Erdős Centennial*, pages 169–264. Springer, Berlin, Heidelberg, 2013.
24. J. A. Garay, P. MacKenzie, and K. Yang. Efficient and universally composable committed oblivious transfer and applications. In *Theory of Cryptography*, pages 297–316, 2004.
25. C. Gentry, P. MacKenzie, and Z. Ramzan. A Method for Making Password-Based Key Exchange Resilient to Server Compromise. In C. Dwork, editor, *Advances in Cryptology - CRYPTO 2006*, pages 142–159, Berlin, Heidelberg, 2006. Springer.
26. O. Goldreich and Y. Lindell. Session-key generation using human passwords only. In J. Kilian, editor, *CRYPTO 2001*, volume 2139 of *LNCS*, pages 408–432. Springer, Berlin, Heidelberg, Aug. 2001.
27. O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In A. Aho, editor, *19th ACM STOC*, pages 218–229. ACM Press, May 1987.
28. Y. Gu, S. Jarecki, P. Kedzior, P. Nazarian, and J. Xu. Threshold PAKE with security against compromise of all servers. In K.-M. Chung and Y. Sasaki, editors, *Advances in Cryptology – ASIACRYPT 2024*, pages 66–100, Singapore, 2025. Springer Nature Singapore.
29. Y. Gu, S. Jarecki, and H. Krawczyk. KHAPE: asymmetric PAKE from key-hiding key exchange. In *Advances in Cryptology - CRYPTO 2021*, pages 701–730, 2021.
30. B. Haase and B. Labrique. AuCPace: Efficient verifier-based PAKE protocol tailored for the IIoT. *IACR TCHES*, 2019(2):1–48, 2019.
31. D. P. Jablon. Extended password key exchange protocols immune to dictionary attacks. In *6th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE 1997)*, pages 248–255, Cambridge, MA, USA, June 18–20, 1997. IEEE Computer Society.
32. S. Jarecki, H. Krawczyk, and J. Xu. OPAQUE: An asymmetric PAKE protocol secure against pre-computation attacks. In J. B. Nielsen and V. Rijmen, editors, *EUROCRYPT 2018, Part III*, volume 10822 of *LNCS*, pages 456–486. Springer, Cham, Apr. / May 2018.
33. S. Jarecki and V. Shmatikov. Efficient Two-Party Secure Computation on Committed Inputs. In M. Naor, editor, *Advances in Cryptology - EUROCRYPT 2007*, pages 97–114, Berlin, Heidelberg, 2007. Springer.

34. J. Katz, R. Ostrovsky, and M. Yung. Efficient password-authenticated key exchange using human-memorable passwords. In B. Pfitzmann, editor, *EUROCRYPT 2001*, volume 2045 of *LNCS*, pages 475–494. Springer, Berlin, Heidelberg, May 2001.
35. S. Kim, H. Lee, and J. H. Seo. Efficient zero-knowledge arguments in discrete logarithm setting: Sublogarithmic proof or sublinear verifier. In S. Agrawal and D. Lin, editors, *Advances in Cryptology – ASIACRYPT 2022*, pages 403–433, 2022.
36. P. D. MacKenzie, T. Shrimpton, and M. Jakobsson. Threshold password-authenticated key exchange. In M. Yung, editor, *CRYPTO 2002*, volume 2442 of *LNCS*, pages 385–400. Springer, Berlin, Heidelberg, Aug. 2002.
37. I. McQuoid, M. Rosulek, and J. Xu. How to obfuscate MPC inputs. In E. Kiltz and V. Vaikuntanathan, editors, *Theory of Cryptography*, pages 151–180, 2022.
38. I. McQuoid and J. Xu. An efficient strong asymmetric PAKE compiler instantiable from group actions. In *ASIACRYPT 2023*, page 176–207, 2023.
39. M. Naor and O. Reingold. Number-theoretic constructions of efficient pseudo-random functions. *J. ACM*, 51(2):231–262, Mar. 2004.
40. R. Radwan and S. Zejnilovic. Password reuse is rampant: nearly half of observed user logins are compromised, 2025. `https://blog.cloudflare.com/password-reuse-rampant-half-user-logins-compromised/`.
41. J. T. Schwartz. Fast probabilistic algorithms for verification of polynomial identities. *J. ACM*, 27(4):701–717, Oct. 1980.
42. M. Shirvanian, S. Jarecki, N. Saxena, and N. Nathan. Two-factor authentication resilient to server compromise using mix-bandwidth devices. In *21st Annual Network and Distributed System Security Symposium, NDSS*, 2014.
43. V. Shoup. Lower bounds for discrete logarithms and related problems. In W. Fumy, editor, *Advances in Cryptology — EUROCRYPT '97*, pages 256–266, 1997.
44. M. Stadler. Publicly verifiable secret sharing. In U. Maurer, editor, *Advances in Cryptology — EUROCRYPT '96*, pages 190–199, Berlin, Heidelberg, 1996. Springer Berlin Heidelberg.
45. A. C.-C. Yao. Protocols for secure computations (extended abstract). In *23rd FOCS*, pages 160–164. IEEE Computer Society Press, Nov. 1982.

A Additional Preliminaries

In this section we include standard definitions deferred from Section 2 due to space constraints.

Hardness Assumptions. We recall the Discrete Logarithm and Decisional Diffie-Hellman assumptions.

Definition 6 (Discrete Logarithm Problem). *Let $\mathbb{G}$ be a cyclic group of prime order p with generator g. The* Discrete Logarithm problem (DL) *on $\mathbb{G}$ is, given (g, g^{ξ}) where $\xi \leftarrow_{\$} \mathbb{Z}_p$, to find ξ. We say that DL is hard in $\mathbb{G}$ if all algorithms running in polynomial time have no better than negligible probability of solving the DL problem.*

Definition 7 (Decisional Diffie-Hellman Problem). *Let $\mathbb{G}$ be a cyclic group of prime order p with generator g. The* Decisional Diffie-Hellman problem (DDH)

on $\mathbb{G}$ is to distinguish Diffie-Hellman tuples from random tuples. In particular, the DDH advantage of an algorithm $\mathcal{A}$ is defined as:

$$\left| \Pr_{a,b \leftarrow_\$ \mathbb{Z}_p}[\mathcal{A}(g, g^a, g^b, g^{ab}) = 1] - \Pr_{a,b,c \leftarrow_\$ \mathbb{Z}_p}[\mathcal{A}(g, g^a, g^b, g^c) = 1] \right|$$

We say that DDH is hard in $\mathbb{G}$ if all algorithms running in polynomial time have no better than negligible DDH advantage.

The hardness of DDH in $\mathbb{G}$ implies the hardness of DL in $\mathbb{G}$.

Commitment Hiding and Binding. As mentioned in Section 2, a secure commitment scheme needs to be hiding and binding, which we formally define below. Both definitions consider an adversary whose implicit input is a security parameter, which is also an implicit input of algorithm COM and an index that defines message, decommitment, and commitment spaces $\mathcal{M}$, $\mathcal{D}$, and $\mathcal{C}$. As a consequence, the adversarial advantage expressions defined below are also functions of the security parameter.

Definition 8 (Commitment Hiding). *Let $\mathsf{COM} : \mathcal{M} \times \mathcal{D} \to \mathcal{C}$ be a commitment scheme. Suppose $\mathcal{A}$ is an algorithm that first outputs two messages $m_0, m_1 \in \mathcal{M}$. The advantage of $\mathcal{A}$ against the hiding property of COM is defined as:*

$$\mathsf{Adv}^{\mathsf{hid}}_{\mathsf{COM}}[\mathcal{A}] = \left| \Pr_{(C,d) \leftarrow \mathsf{COM}(m_0)}[\mathcal{A}(C) = 1] - \Pr_{(C,d) \leftarrow \mathsf{COM}(m_1)}[\mathcal{A}(C) = 1] \right|$$

We say that COM is hiding *if $\mathsf{Adv}^{\mathsf{hid}}_{\mathsf{COM}}[\mathcal{A}]$ is negligible for all efficient $\mathcal{A}$.*

Definition 9 (Commitment Binding). *Let $\mathsf{COM} : \mathcal{M} \times \mathcal{D} \to \mathcal{C}$ be a commitment scheme. The advantage of algorithm $\mathcal{A}$ against the binding property of COM, denoted $\mathsf{Adv}^{\mathsf{bind}}_{\mathsf{COM}}[\mathcal{A}]$, is defined as the probability that $\mathcal{A}$ finds $m_0, m_1 \in \mathcal{M}$ and $d_0, d_1 \in \mathcal{D}$ such that $m_0 \neq m_1$ and $\mathsf{COM}(m_0; d_0) = \mathsf{COM}(m_1; d_1)$. We say that COM is* binding *if $\mathsf{Adv}^{\mathsf{bind}}_{\mathsf{COM}}[\mathcal{A}]$ is negligible for all efficient $\mathcal{A}$.*

ElGamal Commitments. We define ElGamal commitment scheme $\mathsf{ElG.Commit}$ with message space $\mathcal{M} = \mathbb{G}$, randomness space $\mathcal{D} = \mathbb{Z}_p^2$, and commitment space $\mathcal{C} = \mathbb{G}^3$, where $\mathbb{G}$ is a group of prime order p with generator g. The commitment algorithm is:

$$\mathsf{ElG.Commit}(m; d_1, d_2) := (g^{d_1}, g^{d_2}, m \cdot g^{d_1 d_2})$$

$\mathsf{ElG.Commit}$ is perfectly binding and computationally hiding under the DDH assumption. Where the meaning is clear from context, we sometimes abbreviate $\mathsf{ElG.Commit}$ as simply ElG.

Pedersen Commitments. We define ("compacted") Pedersen commitment scheme Ped^N with message space $\mathcal{M} = \mathbb{Z}_p^N$, randomness space $\mathcal{D} = \mathbb{Z}_p$, and commitment space $\mathcal{C} = \mathbb{G}$, where $\mathbb{G}$ is a group of prime order p with generators $g, h_1, \ldots h_N$. Security is based on the hardness of computing a DL between

bases $g, h_1, \ldots h_N$, which can be assumed if, for example, they are provided by a common reference string. Our protocols all use the random oracle model, so $h_1, \ldots, h_N$ could be outputs of a random oracle. The commitment algorithm is:

$$\mathsf{Ped}^N(m_1, ..., m_N; d) := g^d \cdot (h_1)^{m_1} \cdot ... \cdot (h_N)^{m_N}$$

Ped^N is perfectly hiding and computationally binding under the DL assumption.

Committed Oblivious Transfer. Figure 9 is $\mathcal{F}_{\mathsf{COT}}$, an ideal functionality for committed oblivious transfer (COT) with commitment scheme COM. COT, introduced in [17], is a strengthened variant of standard oblivious transfer that allows parties to prove correct behavior relative to committed values. In particular, the receiver learns commitments to both sender messages but only learns the corresponding decommitment to one of them.

The functionality interacts with set of honest parties $\mathcal{P}$ and ideal adversary $\mathcal{A}^*$. The functionality is also parameterized by a global *session identifier*, sid, which we omit from the syntax below for visual clarity.

- On $(\textsc{Send}, \mathsf{ssid_S}, m_0, d_0, m_1, d_1)$ from $\mathsf{S} \in \mathcal{P} \cup \{\mathcal{A}^*\}$ for $m_0, m_1 \in \mathcal{M}$ and $d_0, d_1 \in \mathcal{D}$: If there is no prior record $(\textsc{Send}, \mathsf{ssid_S}, ...)$ then save $(\textsc{Send}, \mathsf{ssid_S}, \mathsf{S}, m_0, d_0, m_1, d_1)$ marked fresh and send $(\textsc{Send}, \mathsf{ssid_S}, \mathsf{S}, \mathsf{COM}(m_0; d_0), \mathsf{COM}(m_1; d_1))$ to $\mathcal{A}^*$.
- On $(\textsc{Rcv}, \mathsf{ssid_R}, \sigma)$ from $\mathsf{R} \in \mathcal{P} \cup \{\mathcal{A}^*\}$ for $\sigma \in \{0, 1\}$: If there is no prior record $(\textsc{Rcv}, \mathsf{ssid_R}, ...)$ then save $(\textsc{Rcv}, \mathsf{ssid_R}, \mathsf{R}, \sigma)$ marked fresh and send $(\textsc{Rcv}, \mathsf{ssid_R}, \mathsf{R})$ to $\mathcal{A}^*$.
- On $(\textsc{Complete}, \mathsf{ssid_S}, \mathsf{ssid_R})$ from $\mathcal{A}^*$: If there are records $(\textsc{Send}, \mathsf{ssid_S}, \mathsf{S}, m_0, d_0, m_1, d_1)$ and $(\textsc{Rcv}, \mathsf{ssid_R}, \mathsf{R}, \sigma)$, both marked fresh, then mark them both $\mathsf{completed}$, and output $(\textsc{Complete}, \mathsf{ssid_R}, m_\sigma, d_\sigma, \mathsf{COM}(m_{1-\sigma}; d_{1-\sigma}))$ to R.

Fig. 9. $\mathcal{F}_{\mathsf{COT}}[\mathsf{COM}]$: Committed oblivious transfer functionality

One of our two TFA-KE protocols ($\mathsf{CKEM}_{\mathsf{PoNR}}$, see Section 5.4) utilizes a COT protocol for commitment scheme Ped^N. There are several COT protocols [11,24,33] in the literature that seem suitable. The latter two are proven UC-secure, but for a model that differs somewhat from our $\mathcal{F}_{\mathsf{COT}}$ in that it does not consider insecure channels and it does not expose commitments and decommitments for use by higher-level applications. We believe that these COT protocols could be proven UC-secure in our model with only small adjustments, but a detailed re-analysis of these protocols is outside the scope of our work.

B Proof of f_{PoNR} One-Wayness and Tightness

In this section we prove Lemma 3, Theorem 5, and Theorem 6, which state, respectively, that f^*_{PoNR} is collision-resistant, f_{PoNR} realizes $\mathcal{F}_{\mathsf{OWF}}$, and f_{PoNR} is tight.

B.1 Proof of Lemma 3

Proof. To prove Lemma 3, first suppose there exists an algorithm $\mathcal{A}$ that outputs $(\mathsf{hpw}, s) \neq (\mathsf{hpw}', s')$ and sid s.t. $f^*_{\mathsf{PoNR}}(\mathsf{sid}, \mathsf{hpw}, s) = f^*_{\mathsf{PoNR}}(\mathsf{sid}, \mathsf{hpw}', s')$ and $\exists_{i\in[N]}\ \mathsf{hpw}_i \cdot \prod_{j\in[t],s[j]=1} \mathsf{H}_{\mathsf{x}}(i,j) \neq \mathsf{hpw}'_i \cdot \prod_{j\in[t],s'[j]=1} \mathsf{H}_{\mathsf{x}}(i,j) \vee \mathsf{hpw}_i \neq \mathsf{hpw}'_i$. As in the proof of Lemma 1, there is a straightforward reduction from $\mathcal{A}$ that solves DL in $\mathbb{G}$.

Here we see the usefulness of the $\hat{\mathsf{H}}_{\mathbb{G}}$ bases, as they ensure that collision is only possible for $\mathsf{hpw} = \mathsf{hpw}'$ and $s \neq s'$ s.t. $\forall_{i\in[N]} \prod_{j\in[t],s[j]=1} \mathsf{H}_{\mathsf{x}}(i,j) = \prod_{j\in[t],s'[j]=1} \mathsf{H}_{\mathsf{x}}(i,j)$. We will show that such $s, s' \in \{0,1\}^t$ exist with negligible probability over the randomness of H_{x}. We abbreviate $\mathsf{H}_{\mathsf{x}}(i,j)$ by $x_{i,j}$.

$$
\begin{aligned}
& \Pr_{\substack{x_{1,1},\ldots,x_{1,t}\leftarrow_{\$}\mathbb{Z}_p^* \\ \cdots \\ x_{N,1},\ldots,x_{N,t}\leftarrow_{\$}\mathbb{Z}_p^*}} \left[\bigcup_{\substack{s,s'\in\{0,1\}^t \\ s\neq s'}} \bigcap_{i\in[N]} \prod_{\substack{j\in[t] \\ s[j]=1}} x_{i,j} = \prod_{\substack{j\in[t] \\ s'[j]=1}} x_{i,j} \right] \\
\leq & \sum_{\substack{s,s'\in\{0,1\}^t \\ s\neq s'}} \Pr_{\substack{x_{1,1},\ldots,x_{1,t}\leftarrow_{\$}\mathbb{Z}_p^* \\ \cdots \\ x_{N,1},\ldots,x_{N,t}\leftarrow_{\$}\mathbb{Z}_p^*}} \left[\bigcap_{i\in[N]} \prod_{\substack{j\in[t] \\ s[j]=1}} x_{i,j} = \prod_{\substack{j\in[t] \\ s'[j]=1}} x_{i,j} \right] \\
= & \sum_{\substack{s,s'\in\{0,1\}^t \\ s\neq s'}} \prod_{i\in[N]} \Pr_{x_{i,1},\ldots,x_{i,t}\leftarrow_{\$}\mathbb{Z}_p^*} \left[\prod_{\substack{j\in[t] \\ s[j]=1}} x_{i,j} = \prod_{\substack{j\in[t] \\ s'[j]=1}} x_{i,j} \right] \\
= & \sum_{\substack{s,s'\in\{0,1\}^t \\ s\neq s'}} \left(\Pr_{x_{1,1},\ldots,x_{1,t}\leftarrow_{\$}\mathbb{Z}_p^*} \left[\prod_{\substack{j\in[t] \\ s[j]=1}} x_{1,j} = \prod_{\substack{j\in[t] \\ s'[j]=1}} x_{1,j} \right] \right)^N \\
= & \sum_{\substack{s,s'\in\{0,1\}^t \\ s\neq s'}} \left(\frac{1}{p-1} \right)^N \\
< & \frac{4^t}{(p-1)^N} \qquad (6)
\end{aligned}
$$

At the fourth line of the above, observe that, since $s \neq s'$, there must be at least one $x_{1,j}$ variable that appears on only one side of the equation. That variable can be isolated to show that it has exactly one possible value in $\mathbb{Z}_p^*$ that satisfies the equation.

To ensure that collisions are negligibly probable, N must be sufficiently large. In particular, by upper-bounding Expression 6 with $2^{-\kappa}$, we obtain the bound $N \geq \frac{\kappa+2t}{\log(p-1)}$ and complete the proof of Lemma 3. Note that for all parameters

we consider reasonable (e.g. $\kappa = 128$, $t = 30$, and $\log p = 256$) any $N \geq 1$ suffices.

B.2 Proof of Theorem 5

Proof. To prove the security of f_{PoNR} we provide in Figure 10 the simulator $\mathsf{SIM}_{\mathsf{PoNR}}$, which operates in the GGM and ROM. $\mathsf{SIM}_{\mathsf{PoNR}}$ follows the same simulation strategy as f_{Vand}'s $\mathsf{SIM}_{\mathsf{Vand}}$. They differ only in that $\mathsf{SIM}_{\mathsf{PoNR}}$ has the additional random oracles $\hat{\mathsf{H}}_{\mathbb{G}}$ and H_{x} to program, and that the $\hat{\mathsf{H}}_{\mathbb{G}}$ bases participate in $\mathsf{SIM}_{\mathsf{PoNR}}$'s internal $\mathbb{G}$ element descriptions. The two simulators' most important difference is in their underlying definitions of $k_i(\mathsf{pw}, s)$.

Like $\mathsf{SIM}_{\mathsf{Vand}}$, $\mathsf{SIM}_{\mathsf{PoNR}}$ perfectly emulates the real world except in the case of a discrete logarithm break, an H_{p} premonition, or a OWF collision. The first event is negligibly probable in the GGM by the same argument as in the proof of Theorem 1. The second event is also negligibly probable under the assumption that ${q_{\mathsf{H}} q_{\mathbb{G}}}^2 \leq 2^{N \log p - \kappa}$. By Lemma 3, in the case that no DL break occurs, the third event (i.e. $f_{\mathsf{PoNR}} = f^*_{\mathsf{PoNR}} \circ f_{\mathsf{pre}}$ collision) is only possible in the case of f_{pre} collision. Under the assumption that at most $2^{(N \log p - \kappa)/2}$ H_{p} queries are made, the probability of such a collision is $< 2^{-\kappa}$ by a simple birthday bound.

B.3 Proof of Theorem 6

Proof. We follow the same proof structure as in the proof of Theorem 2, recalling that f_{Vand} and f_{PoNR} (and $\mathsf{SIM}_{\mathsf{Vand}}$ and $\mathsf{SIM}_{\mathsf{PoNR}}$) differ only in their definitions of $k_i(\mathsf{pw}, s)$ and by the presence of the $\hat{\mathsf{H}}_{\mathbb{G}}$ bases. The proof of Lemma 2 does not depend upon the definition of $k_i(\mathsf{pw}, s)$, nor is it affected by the presence of the $\hat{\mathsf{H}}_{\mathbb{G}}$ bases; therefore its statement also applies to $\mathsf{SIM}_{\mathsf{PoNR}}$.

Lemma 5. *For every integer $c \geq 3$, if the* AdvTest *queries of $\mathsf{SIM}_{\mathsf{PoNR}}$ imply an attack graph $\mathcal{G}$ containing a cycle of length $\leq c$, then there exist distinct $(\mathsf{pw}_1, s_1), \ldots, (\mathsf{pw}_c, s_c) \in \{0,1\}^* \times \{0,1\}^t$ such that $\mathsf{rank}(K((\mathsf{pw}_1, s_1), \ldots, (\mathsf{pw}_c, s_c))) < c$.*

In the case of f_{Vand}, the Vandermonde structure of matrix K made it easy to observe that it could not have rank $< c$ if $c \leq N$. For f_{PoNR}, K is not a Vandermonde matrix, but we will still argue that it is unlikely for a K of low rank to exist.

Lemma 6. *If $c - 1 + \frac{\kappa + c(t + \log q_{\mathsf{H}})}{\log(p-1) - \log(2c(t+1))} \leq N \leq 2c$, then the probability that there exist distinct $(\mathsf{pw}_1, s_1), \ldots, (\mathsf{pw}_c, s_c) \in \{0,1\}^t$ such that $\mathsf{rank}(K((\mathsf{pw}_1, s_1), \ldots, (\mathsf{pw}_c, s_c))) < c$ is at most $2^{-\kappa}$.*

Fix any distinct $(\mathsf{pw}_1, s_1), \ldots, (\mathsf{pw}_c, s_c)$. For all $1 \leq i \leq N$, define K_i to be the submatrix of $K((\mathsf{pw}_1, s_1), \ldots, (\mathsf{pw}_c, s_c))$ consisting of its first i rows. $0 \leq \mathsf{rank}(K_i) \leq \min\{i, c\}$. Denote by LC_i the event that the ith row of K is some linear combination of its first $i-1$ rows. If $\mathsf{rank}(K_i) = \nu$, then either $\mathsf{rank}(K_{i-1}) = \nu - 1$ and LC_i does not occur, or $\mathsf{rank}(K_{i-1}) = \nu$ and LC_i does occur. We

Whenever a new $\mathbb{G}$ element is sent to the environment, the simulator picks a unique random string to represent it (except when this behavior is overridden by the code below). The simulator maintains internal descriptions for these $\mathbb{G}$ elements in the form of three tables of maps α, $\hat{\alpha}$, and β. $\alpha[\gamma]$ and $\hat{\alpha}[\gamma]$ are maps $(\mathsf{sid}, i) \mapsto \mathbb{Z}_p$ and $\beta[\gamma]$ is a map $\mathsf{sid} \mapsto \mathbb{Z}_p$ such that $\gamma \in \mathbb{G}$ is described as:

$$\gamma = \prod_{\mathsf{sid},i} \mathsf{H}_{\mathbb{G}}(\mathsf{sid}, i)^{\alpha[\gamma](\mathsf{sid},i)} \cdot \hat{\mathsf{H}}_{\mathbb{G}}(\mathsf{sid}, i)^{\hat{\alpha}[\gamma](\mathsf{sid},i)} \cdot \prod_{\mathsf{sid}} (y_{\mathsf{sid}})^{\beta[\gamma](\mathsf{sid})} \tag{7}$$

The simulator also maintains a table $X : (\mathsf{sid}, \mathsf{pw}, s) \mapsto \{0, 1\}$ of possible OWF inversions that are known to be either correct or wrong.

- On fresh query $\mathsf{H}_{\mathbb{G}}(\mathsf{sid}, i)$, define new $\gamma \in \mathbb{G}$ and return it. $\alpha[\gamma](\mathsf{sid}, i) := 1$.
- On fresh query $\hat{\mathsf{H}}_{\mathbb{G}}(\mathsf{sid}, i)$, define new $\mathbb{G}$ element γ and return it. $\hat{\alpha}[\gamma](\mathsf{sid}, i) := 1$.
- Sample $\mathsf{H}_{\mathsf{x}}(i, j)$ uniformly at random for every $(i, j) \in [N] \times [t]$.
- On (LeakVal, sid), define new $y_{\mathsf{sid}} \in \mathbb{G}$ and send it to $\mathcal{A}^*$. $\beta[y_{\mathsf{sid}}](\mathsf{sid}) := 1$.
- On group operation $\gamma_1 \cdot \gamma_2$, define new $\mathbb{G}$ element γ_3 and return it. $\alpha[\gamma_3] := \alpha[\gamma_1] + \alpha[\gamma_2]$, $\hat{\alpha}[\gamma_3] := \hat{\alpha}[\gamma_1] + \hat{\alpha}[\gamma_2]$, and $\beta[\gamma_3] := \beta[\gamma_1] + \beta[\gamma_2]$.

In the above, unspecified values of the α and β maps default to 0, and $+$ represents component-wise addition of the full maps, including all sid's.

Whenever a new $\mathbb{G}$ element γ is defined (by $\mathsf{H}_{\mathbb{G}}$ or $\hat{\mathsf{H}}_{\mathbb{G}}$ query, LeakVal event, or group operation), the simulator checks whether its internal description is identical to any previous γ'; if so, the same unique string is used as its representation. If not, the simulator proceeds to check whether the internal description of γ is *conditionally* equivalent to any previous γ', conditioned on possible OWF inversions:

$$\begin{aligned}
&\forall_{\mathsf{sid}\ \text{s.t.}\ \beta[\gamma](\mathsf{sid})=\beta[\gamma'](\mathsf{sid})} \\
&\quad \forall_{i\in[N]} \quad \alpha[\gamma](\mathsf{sid}, i) = \alpha[\gamma'](\mathsf{sid}, i) \wedge \hat{\alpha}[\gamma](\mathsf{sid}, i) = \hat{\alpha}[\gamma'](\mathsf{sid}, i) \\
&\qquad\qquad \wedge \\
&\forall_{\mathsf{sid}\ \text{s.t.}\ \beta[\gamma](\mathsf{sid})\neq\beta[\gamma'](\mathsf{sid})} \quad \exists_{\mathsf{pw},s} \\
&\quad \forall_{i\in[N]} \quad \alpha[\gamma](\mathsf{sid}, i) - \alpha[\gamma'](\mathsf{sid}, i) = k_i(\mathsf{pw}, s) \cdot (\beta[\gamma](\mathsf{sid}) - \beta[\gamma'](\mathsf{sid})) \\
&\qquad\qquad\qquad\qquad \wedge \\
&\qquad\qquad \hat{\alpha}[\gamma](\mathsf{sid}, i) - \hat{\alpha}[\gamma'](\mathsf{sid}, i) = \mathsf{H}_{\mathsf{p}}(i, \mathsf{pw}) \cdot (\beta[\gamma](\mathsf{sid}) - \beta[\gamma'](\mathsf{sid}))
\end{aligned} \tag{8}$$

If Condition 8 is true with some $(\mathsf{sid}_1, \mathsf{pw}_1, s_1), (\mathsf{sid}_2, \mathsf{pw}_2, s_2), \ldots, (\mathsf{sid}_n, \mathsf{pw}_n, s_n)$ satisfying the second clause, then consider each of those as a possible OWF inversion. If all these inversions are correct, then $\gamma = \gamma'$; otherwise, $\gamma \neq \gamma'$. If there is insufficient information in X to determine whether $\gamma = \gamma'$, perform the relevant (AdvTest, sid, (pw, s)) queries until receiving one "wrong" or receiving all "correct". Record responses in X to avoid redundant queries in the future.

Fig. 10. $\mathsf{SIM}_{\mathsf{PoNR}}$: Simulator for f_{PoNR}, with differences from $\mathsf{SIM}_{\mathsf{Vand}}$ in shadowed text

therefore have the following recurrence relation for $1 \leq i \leq N$ and $0 \leq \nu \leq \min\{i, c-1\}$:

$$\begin{aligned}\Pr[\mathsf{rank}(K_i) = \nu] \leq\ & \Pr[\mathsf{rank}(K_{i-1}) = \nu - 1] \\ & + \Pr[\mathsf{rank}(K_{i-1}) = \nu] \cdot \Pr[LC_i \mid \mathsf{rank}(K_{i-1}) = \nu]\end{aligned} \tag{9}$$

Conditioned on the event that $\mathsf{rank}(K_{i-1}) = \nu < c$, we can define the square matrix S_i which consists of a basis of the row space of K_{i-1} augmented with the ith row of K, all truncated to just the first $\nu + 1$ columns. For event LC_i to occur, it is necessary (though not sufficient) that $\det(S_i) = 0$. We can therefore bound the probability of event LC_i. We abbreviate $\mathsf{H_p}(i, \mathsf{pw}_\ell)$ as $hpw_{i,\ell}$, and we abbreviate $\mathsf{H_x}(i, j)$ as $x_{i,j}$. All of the following probabilities are conditioned on the event $\mathsf{rank}(K_{i-1}) = \nu$, which for brevity we do not repeat on every line.

$$\begin{aligned}
& \Pr_{\substack{hpw_{1,1},\ldots,hpw_{1,c},\\ x_{1,1},\ldots,x_{1,t} \leftarrow_\$ \mathbb{Z}_p^* \\ \cdots \\ hpw_{i,1},\ldots,hpw_{i,c},\\ x_{i,1},\ldots,x_{i,t} \leftarrow_\$ \mathbb{Z}_p^*}} [LC_i \mid \mathsf{rank}(K_{i-1}) = \nu] \\
\leq\ & \Pr_{\substack{hpw_{1,1},\ldots,hpw_{1,c},\\ x_{1,1},\ldots,x_{1,t} \leftarrow_\$ \mathbb{Z}_p^* \\ \cdots \\ hpw_{i,1},\ldots,hpw_{i,c},\\ x_{i,1},\ldots,x_{i,t} \leftarrow_\$ \mathbb{Z}_p^*}} [0 = \det(S_i)] \\
=\ & \Pr_{\substack{hpw_{1,1},\ldots,hpw_{1,c},\\ x_{1,1},\ldots,x_{1,t} \leftarrow_\$ \mathbb{Z}_p^* \\ \cdots \\ hpw_{i,1},\ldots,hpw_{i,c},\\ x_{i,1},\ldots,x_{i,t} \leftarrow_\$ \mathbb{Z}_p^*}} \left[0 = \sum_{\ell \in [\nu+1]} C_{\nu+1,\ell}(S_i) \cdot k_i(\mathsf{pw}_\ell, s_\ell)\right] \\
=\ & \Pr_{\substack{hpw_{1,1},\ldots,hpw_{1,c},\\ x_{1,1},\ldots,x_{1,t} \leftarrow_\$ \mathbb{Z}_p^* \\ \cdots \\ hpw_{i,1},\ldots,hpw_{i,c},\\ x_{i,1},\ldots,x_{i,t} \leftarrow_\$ \mathbb{Z}_p^*}} \left[0 = \sum_{\ell \in [\nu+1]} C_{\nu+1,\ell}(S_i) \cdot hpw_{i,\ell} \cdot \prod_{\substack{j \in [t] \\ s_\ell[j]=1}} x_{i,j}\right] \\
\leq\ & \frac{t+1}{p-1}
\end{aligned}$$

At line 3 of the above, we use Laplace's formula for the determinant, denoting by $C_{\nu+1,\ell}(S_i)$ the $(\nu+1, \ell)$ cofactor of S_i. Observe that the variables $hpw_{i,\ell}, x_{i,1}, \ldots, x_{i,t}$ do not participate in $C_{\nu+1,\ell}(S_i)$. We can therefore treat the equation in line 4 as some fixed polynomial evaluated on randomly sampled values for those $t+1$ variables. At the last line, we invoke the Schwartz-Zippel lemma [41], which states that a point randomly sampled over $\mathbb{F}' \subseteq \mathbb{F}$ on a non-zero polynomial of degree $\leq t+1$ over finite field $\mathbb{F}$ is zero with probability $\leq \frac{t+1}{|\mathbb{F}'|}$. The first ν rows of S_i are by definition linearly independent, which ensures that the cofactors are all non-zero. The fact that $(\mathsf{pw}_1, s_1), \ldots, (\mathsf{pw}_c, s_c)$ are distinct therefore guarantees that the polynomial in question is non-zero.

Returning with this bound to Inequality 9, we have:

$$\Pr[\mathsf{rank}(K_i) = \nu] \leq \Pr[\mathsf{rank}(K_{i-1}) = \nu - 1] + \Pr[\mathsf{rank}(K_{i-1}) = \nu] \cdot \frac{t+1}{p-1} \quad (10)$$

Along with the base case $\Pr[\mathsf{rank}(K_0) = 0] = 1$, Inequality 10 is a slight generalization of Pascal's recurrence relation for binomial coefficients. It has the following solution (for all $0 \leq i \leq N$ and $0 \leq \nu \leq \min\{i, c-1\}$):

$$\Pr[\mathsf{rank}(K_i) = \nu] \leq \binom{i}{\nu} \cdot \left(\frac{t+1}{p-1}\right)^{i-\nu} \quad (11)$$

Setting $i = N$, we can finally show that K $(= K_N)$ is unlikely to be of low rank.

$$\begin{aligned}
\Pr[\mathsf{rank}(K) < c] &\leq \sum_{\nu=0}^{c-1} \binom{N}{\nu} \cdot \left(\frac{t+1}{p-1}\right)^{N-\nu} \\
&\leq \binom{N}{c} \cdot \left(\frac{t+1}{p-1}\right)^{N-c} \cdot \sum_{\nu=0}^{c-1} \binom{c}{\nu} \cdot \left(\frac{t+1}{p-1}\right)^{c-\nu} \\
&< N^{N-c} \cdot \left(\frac{t+1}{p-1}\right)^{N-c} \cdot \left[2 \cdot c \cdot \frac{t+1}{p-1}\right] \\
&= \frac{2c(t+1)}{p-1} \cdot \left(\frac{N(t+1)}{p-1}\right)^{N-c}
\end{aligned}$$

Even for $c = N$ (the largest c in consideration), the above probability is negligible. However, this bound only applies to some fixed $(\mathsf{pw}_1, s_1), \ldots, (\mathsf{pw}_c, s_c)$. We use a union bound to show that the event is negligibly probable for any $(\mathsf{pw}_1, s_1), \ldots, (\mathsf{pw}_c, s_c)$.

$$\begin{aligned}
& \Pr\left[\bigcup_{\substack{(\mathsf{pw}_1,s_1),\ldots,(\mathsf{pw}_c,s_c)\in\{0,1\}^*\times\{0,1\}^t \\ \text{all distinct}} \mathsf{rank}(K((\mathsf{pw}_1, s_1), \ldots, (\mathsf{pw}_c, s_c))) < c\right] \\
\leq & \sum_{\substack{(\mathsf{pw}_1,s_1),\ldots,(\mathsf{pw}_c,s_c)\in\{0,1\}^*\times\{0,1\}^t \\ \text{all distinct}}} \Pr[\mathsf{rank}(K((\mathsf{pw}_1, s_1), \ldots, (\mathsf{pw}_c, s_c))) < c] \\
< & (q_{\mathsf{H}} \cdot 2^t)^c \cdot \frac{2c(t+1)}{p-1} \cdot \left(\frac{N(t+1)}{p-1}\right)^{N-c} \quad (12)
\end{aligned}$$

To ensure this probability is negligible, we must set parameter N somewhat greater than c. Assuming as in the statement of Lemma 6 that $N \leq 2c$ (which we will see to be sufficient for reasonable parameters), and upper-bounding Expression 12 with $2^{-\kappa}$, we reach the result needed to complete the proof of Lemma 6.

$$N \geq c - 1 + \frac{\kappa + c(t + \log q_{\mathsf{H}})}{\log(p-1) - \log(2c(t+1))}$$

For example, if $\kappa = 128$, $t = 30$, $\log p = 256$, and $\log q_{\mathsf{H}} = 128$, then this condition is approximated by $N \geq 1.66c$. Then, to ensure that the attack graph $\mathcal{G}$ contains no cycles of length c or less, one could set $N \approx 1.66c$.

To complete the proof of Theorem 6 we return to Theorems 3 and 4 of extremal graph theory, taking c to be the greatest integer satisfying $c - 1 + \frac{\kappa + c(t + \log q_{\mathsf{H}})}{\log(p-1) - \log(2c(t+1))} \leq N \leq 2c$. Theorem 3 implies that the number of edges in $\mathcal{G}$ is $\mathcal{O}({q_{\mathbb{G}}}^{1+1/\lfloor c/2 \rfloor})$, i.e. $\mathsf{SIM}_{\mathsf{PoNR}}$ has rate $r = 1 + \frac{1}{\lfloor c/2 \rfloor}$. Setting $\delta = 4$, Theorem 4 implies that if $c \geq \log q_{\mathbb{G}}$ then the number of edges in $\mathcal{G}$ is $\mathcal{O}(q_{\mathbb{G}})$, i.e. $\mathsf{SIM}_{\mathsf{PoNR}}$ has rate $r = 1$ and f_{PoNR} is tight. Theorem 6 is proven.

For example, if $\log q_{\mathbb{G}} = 128$ and other parameters are as above, then f_{PoNR} is tight for $N = 211$.

C Proof of Committed f_{Vand} and f_{PoNR} One-Wayness and Tightness

In this section we prove Lemma 4, i.e. that $\mathsf{CommittedOwf}[f_{\mathsf{Vand}}, \mathsf{ElG}]$ and $\mathsf{CommittedOwf}[f_{\mathsf{PoNR}}, \mathsf{ElG}]$ have the same one-wayness and tightness properties as f_{Vand} and f_{PoNR}, respectively.

Proof. We write the proof for f_{Vand}. Exactly the same argument applies to f_{PoNR}.

$\mathsf{SIM}_{\mathsf{Vand}}$ can be modified to simulate $\mathsf{CommittedOwf}[f_{\mathsf{Vand}}, \mathsf{ElG}]$. As defined in Appendix A, ElGamal commitment $\mathsf{ElG}(m) := (g^{d_1}, g^{d_2}, m \cdot g^{d_1 d_2})$ where $d_1, d_2 \leftarrow_{\$} \mathbb{Z}_p$. On $(\textsc{Eval}, \mathsf{sid}, \mathsf{P})$ from $\mathcal{F}_{\mathsf{OWF}}$, $\mathsf{SIM}_{\mathsf{Vand}}$ defines three new $\mathbb{G}$ elements $C_{\mathsf{sid}} = (c_{\mathsf{sid},1}, c_{\mathsf{sid},2}, c_{\mathsf{sid},3})$ and sends them to $\mathcal{A}^*$. $\mathsf{SIM}_{\mathsf{Vand}}$ must incorporate $\mathbb{G}$ generator g and all the $c_{\mathsf{sid},1}$, $c_{\mathsf{sid},2}$, and $c_{\mathsf{sid},3}$ bases in its internal descriptions of $\mathbb{G}$ elements. Condition 2 is augmented so that γ and γ' only constitute a OWF inversion test if the exponents of all of these bases are equal in their descriptions.

On event $(\textsc{LeakVal}, \mathsf{sid})$, $\mathsf{SIM}_{\mathsf{Vand}}$ samples $d_{\mathsf{sid}} = (d_{\mathsf{sid},1}, d_{\mathsf{sid},2}) \leftarrow_{\$} \mathbb{Z}_p^2$ and sends it to $\mathcal{A}^*$ along with y_{sid}. The simulator internally defines $c_{\mathsf{sid},1} = g^{d_{\mathsf{sid},1}}$, $c_{\mathsf{sid},2} = g^{d_{\mathsf{sid},2}}$, $c_{\mathsf{sid},3} = y_{\mathsf{sid}} \cdot g^{d_{\mathsf{sid},1} d_{\mathsf{sid},2}}$ and updates its descriptions of all $\mathbb{G}$ elements accordingly. $c_{\mathsf{sid},1}$, $c_{\mathsf{sid},2}$, and $c_{\mathsf{sid},3}$ are excluded from all future $\mathbb{G}$ element descriptions in favor of g and y_{sid}.

These changes to $\mathsf{SIM}_{\mathsf{Vand}}$ introduce a new possibility of deviation from the real world, if $\mathsf{SIM}_{\mathsf{Vand}}$ programs some $\gamma, \gamma' \in \mathbb{G}$ as if they are unequal and then a later choice of d_{sid} reveals that $\gamma = \gamma'$. Intuitively, an environment that causes such an event must be computing the discrete logarithm of $c_{\mathsf{sid},1}$ or $c_{\mathsf{sid},2}$. Accordingly, the classical GGM proof of Shoup [43] can be adapted to show that the event is negligibly probable. In short, this proof is based on the observation that any γ, γ' pair's probabilty of having been misprogrammed as unequal is at most $1/p$ over the random choice of d_{sid}. If at most $q_{\mathbb{G}}$ $\mathbb{G}$ elements are seen by the environment, then the total probability of a misprogramming is $< q_{\mathbb{G}}^2/p$.

More than one d_{sid} is sampled by $\mathsf{SIM}_{\mathsf{Vand}}$, which we could handle by simply multiplying the above bound by the number of $(\textsc{LeakVal}, \mathsf{sid})$ events. This, however, is unnecessary. If a γ, γ' pair's equality is dependent on the choice of more than one d_{sid}, then their misprogramming is not determined until the last

of those d_{sid}'s is sampled. In other words, every γ, γ' pair only contributes a $1/p$ probability of misprogramming once, and the standard $q_{\mathbb{G}}^2/p$ bound holds.

Thus, $\mathsf{CommittedOwf}[f_{\mathsf{Vand}}, \mathsf{ElG}]$ realizes $\mathcal{F}_{\mathsf{OWF}}$ with the same tightness as f_{Vand}.

D Syntax of a TFA-KE Scheme

In this section we further explain the TFA-KE functionality $\mathcal{F}_{\mathsf{TFA\text{-}KE}}$ of Section 4 by mapping its syntax to a tuple of algorithms and discussing their intuitive properties.

A TFA-KE scheme is a tuple of algorithms $(\mathsf{Init}, \mathsf{AuthCodeGen}, \mathsf{Auth}_{\mathsf{C}}, \mathsf{Auth}_{\mathsf{S}})$, to be executed by three parties denoted server S, user's client machine C, and user's personal device D. The TFA-KE scheme is parametrized by integer t which sets the length of the one-time authentication codes produced by the device D.

Since we will model TFA-KE in the UC setting, the TFA-KE procedures below take additional identifiers as inputs. First, all procedures except for $\mathsf{AuthCodeGen}$ take as an additional input a *session identifier* sid, which denotes a unique identifier of some user's account on some server, the account to which this user authenticates via the TFA-KE scheme. Secondly, the $\mathsf{Auth}_{\mathsf{C}}$ and $\mathsf{Auth}_{\mathsf{S}}$ algorithms, which model respectively the client-side and the server-side online authentication algorithms, take as an additional input a *subsession identifier* ssid, which denotes a unique string identifying this online authentication instance, e.g. a binder to a TLS session on which the client and the server interact. Since the role of these identifiers is explained above, we mark these inputs in gray and omit them from the discussion below.

- $\mathsf{Init}(1^\kappa, t, \mathsf{sid}, \mathsf{pw})$: The initialization algorithm takes the security parameter κ, code length parameter t, and a password pw, and outputs a device state st_{D} and a server state file.
- $\mathsf{AuthCodeGen}(\mathsf{st}_{\mathsf{D}}, T)$: The device algorithm takes as inputs the device state st_{D} and an epoch counter T, and outputs a one-time authenticator $\mathsf{code} \in \{0,1\}^t$. We assume that $\mathsf{AuthCodeGen}$ is a deterministic algorithm, and we denote the function it computes as $\mathsf{R}_{\mathsf{st}_{\mathsf{D}}} : \mathbb{N} \to \{0,1\}^t$.
- $\mathsf{Auth}_{\mathsf{C}}(\mathsf{sid}, \mathsf{ssid}, \mathsf{pw}, \mathsf{code})$ The client algorithm takes as inputs password pw, and code generated by $\mathsf{AuthCodeGen}$. The algorithm outputs a key SK_{C} after an interaction with the server algorithm instance $\mathsf{Auth}_{\mathsf{S}}$.
- $\mathsf{Auth}_{\mathsf{S}}(\mathsf{sid}, \mathsf{ssid}, \mathsf{file}, T)$: The server algorithm takes as inputs the server state file and an epoch counter T. The algorithm outputs a key SK_{S} after an interaction with the client algorithm instance $\mathsf{Auth}_{\mathsf{C}}$.

The scheme is correct if for every $\kappa, t, \mathsf{pw}, T$, if $(\mathsf{st}_{\mathsf{D}}, \mathsf{file}) \leftarrow \mathsf{Init}(1^\kappa, t, \mathsf{pw})$, $\mathsf{code} \leftarrow \mathsf{AuthCodeGen}(\mathsf{st}_{\mathsf{D}}, T)$, and algorithms $\mathsf{Auth}_{\mathsf{C}}(\mathsf{pw}, \mathsf{code})$ and $\mathsf{Auth}_{\mathsf{S}}(\mathsf{file}, T)$ interact without adversarial interference, then $\mathsf{SK}_{\mathsf{C}} = \mathsf{SK}_{\mathsf{S}}$.

Intuitively, a TFA-KE scheme is *online secure* if one can distinguish C's or S's outputs SK_{C} and SK_{S} from random only by authenticating under the correct password pw and the correct t-bit code $\mathsf{code} = \mathsf{R}_{\mathsf{st}_{\mathsf{D}}}(T)$ for epoch T assumed by

the attacked party. To neutralize adversarial lunch-time attacks on the device, an authentication code revealed in epoch T, i.e. $\mathsf{code} = \mathsf{R}_{\mathsf{st_D}}(T)$, should leak no information on $\mathsf{code}' = \mathsf{R}_{\mathsf{st_D}}(T')$ for any $T' \neq T$, which implies that $\mathsf{R}_{\mathsf{st_D}}$ must be a pseudorandom function. A TFA-KE scheme is *offline secure* if, given a user whose password pw is chosen at random from dictionary D, an attacker who compromises server S's private state file but does not receive an authenticator code for any epoch T, can learn pw only via a brute-force search which takes time $\Theta(|D| * 2^t)$. Moreover, an attacker who learns both S's state file and an authenticator one-time code for any epoch T, can recover pw only via a brute-force search in time $\Theta(|D|)$.

E Proof of $\mathsf{TFA}_{\mathsf{gen}}$ Security

In this section we prove Theorem 7, i.e. that protocol $\mathsf{TFA}_{\mathsf{gen}}$ realizes $\mathcal{F}_{\mathsf{TFA\text{-}KE}}$.

Initialization

- On (StoreFile, sid, S) from $\mathcal{F}_{\mathsf{TFA\text{-}KE}}$:
  - Send (Eval, sid, S) to $\mathsf{SIM}_{\mathsf{OWF}}$ as if from $\mathcal{F}_{\mathsf{OWF}}$, receive C_F as response.
  - Send (Prove, S, "init", (sid, C_F)) to $\mathcal{A}^*$ as if from $\mathcal{F}_{\mathsf{NIZK}}$, receive (Proof, "init", π) as response, and save record ((sid, C_F), $\perp$, π).

Server File Compromise, Offline Attacks, Hash queries

- On (LeakVal, sid) from $\mathsf{SIM}_{\mathsf{OWF}}$ [file compromise visible to the environment]:
  - Send (CompromiseFile, sid) to $\mathcal{F}_{\mathsf{TFA\text{-}KE}}$ and mark sid as compromised.
  - (If $\mathsf{SIM}_{\mathsf{OWF}}$ makes AdvTest queries service them as below.)
  - Wait for (F, d_F) from $\mathsf{SIM}_{\mathsf{OWF}}$, pick $v_1 \leftarrow_{\$} \mathbb{Z}_{2^t}$ and $k \leftarrow_{\$} \{0,1\}^{\kappa}$.
  - Abort [*abort.key*] if $\mathcal{A}^*$ made a prior query to $\mathsf{H}^{\mathsf{sid}}_{\mathsf{code}}$ on (k, T) for any T.
  - Reveal file := (k, F, C_F, d_F, π) to $\mathcal{A}^*$.
- On (AdvTest, sid, (pw*, s^*)) from $\mathsf{SIM}_{\mathsf{OWF}}$:
  - Send (OfflineTest, sid, pw*, 1, $s^* - v_1$) to $\mathcal{F}_{\mathsf{TFA\text{-}KE}}$.
  - Forward $\mathcal{F}_{\mathsf{TFA\text{-}KE}}$'s response "correct/wrong (guess)" to $\mathsf{SIM}_{\mathsf{OWF}}$.
- On $\mathcal{A}^*$'s query (k^*, T) to $\mathsf{H}^{\mathsf{sid}}_{\mathsf{code}}$, service it as $\mathcal{F}_{\mathsf{RO}}$ except if sid is compromised and $k^* = k$, in which case:
  - Send (EvalCodeDiff, sid, 1, T) to $\mathcal{F}_{\mathsf{TFA\text{-}KE}}$ and receive Δ_T in response.
  - Program $\mathsf{H}^{\mathsf{sid}}_{\mathsf{code}}(k, T) := v_1 - \Delta_T$.
- On query pw on hash $\mathsf{H_p}$, service it as $\mathcal{F}_{\mathsf{RO}}$, but in addition:
  - Insert $(\mathsf{pw}, \mathsf{H_p}(\mathsf{pw}))$ into (initially empty) table $\mathsf{T_p}$.
  - Abort [*abort.hash*] if $\mathsf{T_p}$ contains $(\mathsf{pw}', \mathsf{H_p}(\mathsf{pw}))$ for any $\mathsf{pw}' \neq \mathsf{pw}$.

Fig. 11. $\mathsf{SIM}_{\mathsf{TFA\text{-}gen}}$: Simulator for protocol $\mathsf{TFA}_{\mathsf{gen}}$, part 1: Initialization, Server Corruption, Offline Attacks, Hash query processing

Proof. The simulator SIM that exhibits the claim that protocol $\mathsf{TFA}_{\mathsf{gen}}$ realizes functionality $\mathcal{F}_{\mathsf{TFA\text{-}KE}}$ is described in Figures 11 and 12. To handle the com-

Authentication

[Below sending command $(\text{RKey}, \mathsf{sid}, \mathsf{ssid}, \mathsf{P})$ to $\mathcal{F}_{\mathsf{TFA\text{-}KE}}$ is a short-cut for sending commands $(\text{TestPwd}, \mathsf{sid}, \mathsf{ssid}, \mathsf{P}, \bot, \bot)$ and $(\text{NewKey}, \mathsf{sid}, \mathsf{ssid}, \mathsf{P}, \bot)$ to $\mathcal{F}_{\mathsf{TFA\text{-}KE}}$.]

- On $(\text{CSession}, \mathsf{sid}, \mathsf{ssid}, \mathsf{C}, \mathsf{S})$ from $\mathcal{F}_{\mathsf{TFA\text{-}KE}}$, service client session $\mathsf{C}^{\mathsf{ssid}}$:
  1. Receive (C'_F, π') from $\mathcal{A}^*$ on behalf of $\mathsf{C}^{\mathsf{ssid}}$.
  2. Set $\mathsf{ckid} := \mathsf{sid} \,|\, \mathsf{ssid}$ and mark session $\mathsf{C}^{\mathsf{ckid}}$ as fresh.
  3. Send $(\text{CEval}, \mathsf{ckid}, \mathsf{C}, (\mathsf{sid}, C'_F))$ as if from $\mathcal{F}_{\mathsf{CKEM}}$ to $\mathcal{A}^*$.
  4. If there is no record $((\mathsf{sid}, C'_F), \cdot, \pi')$ then send $(\text{Verify}, \mathsf{C}, \mathsf{ckid}, (\mathsf{sid}, C'_F), \pi')$ as if from $\mathcal{F}_{\mathsf{OWF}}$ to $\mathcal{A}^*$, receive $(\text{Witness}, \mathsf{ckid}, \mathbb{w})$ from $\mathcal{A}^*$ and:
     (a) If $\mathbb{w} = (\mathsf{hpw}, s, {d_F}')$ s.t. $C'_F = \mathsf{COM}(f^*(\mathsf{sid}, \mathsf{hpw}, s); {d_F}')$ then save record $((\mathsf{sid}, C'_F), (\mathsf{hpw}, s, {d_F}'), \pi')$;
     (b) else skip steps 5,6 below and regardless of $\mathcal{A}^*$'s response on $\mathcal{F}_{\mathsf{CKEM}}$ interface mark $\mathsf{C}^{\mathsf{ssid}}$ used and send $(\text{RKey}, \mathsf{sid}, \mathsf{ssid}, \mathsf{C})$ to $\mathcal{F}_{\mathsf{TFA\text{-}KE}}$.
  5. If $\mathcal{A}^*$ sends $(\text{Cout}, \mathsf{ckid})$ as if to $\mathcal{F}_{\mathsf{CKEM}}$ and $\exists$ session $\mathsf{S}^{\mathsf{ckid}}$ marked $\mathsf{completed}$ then mark $\mathsf{C}^{\mathsf{ckid}}$ used and:
     (a) If $C'_F \neq C_F$ send $(\text{RKey}, \mathsf{sid}, \mathsf{ssid}, \mathsf{C})$ to $\mathcal{F}_{\mathsf{TFA\text{-}KE}}$;
     (b) else send $(\text{NewKey}, \mathsf{sid}, \mathsf{ssid}, \mathsf{C}, \bot)$ to $\mathcal{F}_{\mathsf{TFA\text{-}KE}}$.
  6. If $\mathcal{A}^*$ sends $(\text{AdvSeval}, \mathsf{ckid}, y, \mathsf{SK}^*)$ as if to $\mathcal{F}_{\mathsf{CKEM}}$ then mark $\mathsf{C}^{\mathsf{ckid}}$ used, parse $y = (\mathsf{sid}', F', s_{\mathsf{S}}, {d_F}')$ (if $y = \bot$ then set all fields to $\bot$), and:
     (a) If $(\mathsf{sid}', C'_F) \neq (\mathsf{sid}, \mathsf{COM}(F'; {d_F}'))$ send $(\text{RKey}, \mathsf{sid}, \mathsf{ssid}, \mathsf{C})$ to $\mathcal{F}_{\mathsf{TFA\text{-}KE}}$;
     (b) elseif $C'_F = C_F$ and sid is marked $\mathsf{compromised}$ send $(\text{Impersonate}, \mathsf{sid}, \mathsf{ssid}, 1, v_1 - s_{\mathsf{S}})$, $(\text{NewKey}, \mathsf{sid}, \mathsf{ssid}, \mathsf{C}, \mathsf{SK}^*)$ to $\mathcal{F}_{\mathsf{TFA\text{-}KE}}$;
     (c) elseif $C'_F = C_F$ but sid is not $\mathsf{compromised}$ then Abort $[abort.cmt]$;
     (d) else retrieve record $((\mathsf{sid}, C'_F), \mathbb{w}, \pi')$, parse $\mathbb{w} = (\mathsf{hpw}, s, {d_F}')$, and:
        i. If $\exists \mathsf{pw}$ s.t. $(\mathsf{pw}, \mathsf{hpw}) \in \mathsf{T_p}$ then send $(\text{TestPwd}, \mathsf{sid}, \mathsf{ssid}, \mathsf{C}, \mathsf{pw}, s - s_{\mathsf{S}})$, $(\text{NewKey}, \mathsf{sid}, \mathsf{ssid}, \mathsf{C}, \mathsf{SK}^*)$ to $\mathcal{F}_{\mathsf{TFA\text{-}KE}}$;
        ii. else send $(\text{RKey}, \mathsf{sid}, \mathsf{ssid}, \mathsf{C})$ to $\mathcal{F}_{\mathsf{TFA\text{-}KE}}$.

- On $(\text{SSession}, \mathsf{sid}, \mathsf{ssid}, \mathsf{S}, \mathsf{C})$ from $\mathcal{F}_{\mathsf{TFA\text{-}KE}}$, service server session $\mathsf{S}^{\mathsf{ssid}}$:
  1. Send (C_F, π) to $\mathcal{A}^*$, set $\mathsf{ckid} := \mathsf{sid} \,|\, \mathsf{ssid}$, mark session $\mathsf{S}^{\mathsf{ckid}}$ as fresh.
  2. Send $(\text{Seval}, \mathsf{ckid}, \mathsf{C}, (\mathsf{sid}, C_F))$ as if from $\mathcal{F}_{\mathsf{CKEM}}$ to $\mathcal{A}^*$.
  3. If $\mathcal{A}^*$ sends $(\text{Sout}, \mathsf{ckid})$ as if to $\mathcal{F}_{\mathsf{CKEM}}$ and $\exists$ session $\mathsf{C}^{\mathsf{ckid}}$ marked fresh then mark $\mathsf{S}^{\mathsf{ckid}}$ $\mathsf{completed}$ and send $(\text{NewKey}, \mathsf{sid}, \mathsf{ssid}, \mathsf{S}, \bot)$ to $\mathcal{F}_{\mathsf{TFA\text{-}KE}}$.
  4. If $\mathcal{A}^*$ sends $(\text{AdvCeval}, \mathsf{ckid}, x)$ as if to $\mathcal{F}_{\mathsf{CKEM}}$ then mark $\mathsf{S}^{\mathsf{ckid}}$ used, parse $x = (\mathsf{sid}', \mathsf{hpw}, s_{\mathsf{C}})$ (if $x = \bot$ then set all fields to $\bot$), and:
     (a) If $\mathsf{sid}' \neq \mathsf{sid}$ or $(\cdot, \mathsf{hpw}) \notin \mathsf{T_p}$ then send $(\text{RKey}, \mathsf{sid}, \mathsf{ssid}, \mathsf{C})$ to $\mathcal{F}_{\mathsf{TFA\text{-}KE}}$ and $(\text{Cout}, \mathsf{ssid}, \bot)$ as if from $\mathcal{F}_{\mathsf{CKEM}}$ to $\mathcal{A}^*$;
     (b) else let $(\mathsf{pw}, \mathsf{hpw}) \in \mathsf{T_p}$, send $(\text{TestPwd}, \mathsf{sid}, \mathsf{ssid}, \mathsf{S}, \mathsf{pw}, s_{\mathsf{C}})$ to $\mathcal{F}_{\mathsf{TFA\text{-}KE}}$, and:
        i. if $\mathcal{F}_{\mathsf{TFA\text{-}KE}}$ responds "correct guess" then pick $\mathsf{SK} \leftarrow_{\$} \{0,1\}^{\kappa}$ and send $(\text{NewKey}, \mathsf{sid}, \mathsf{ssid}, \mathsf{S}, \mathsf{SK})$ to $\mathcal{F}_{\mathsf{TFA\text{-}KE}}$ and $(\text{Cout}, \mathsf{ssid}, \mathsf{SK})$ as if from $\mathcal{F}_{\mathsf{CKEM}}$ to $\mathcal{A}^*$;
        ii. else send $(\text{NewKey}, \mathsf{sid}, \mathsf{ssid}, \mathsf{S}, \bot)$ to $\mathcal{F}_{\mathsf{TFA\text{-}KE}}$ and $(\text{Cout}, \mathsf{ssid}, \bot)$ as if from $\mathcal{F}_{\mathsf{CKEM}}$ to $\mathcal{A}^*$.

Fig. 12. $\mathsf{SIM}_{\mathsf{TFA\text{-}gen}}$: Simulator for protocol $\mathsf{TFA}_{\mathsf{gen}}$, part 2: Authentication

promise of the server file, which reveals the tight OWF function value $F = f(\mathsf{sid}, \overline{\mathsf{hpw}}, s)$, and offline dictionary attacks that search for the preimage $(\overline{\mathsf{hpw}}, s)$ given this value, simulator SIM uses as a sub-procedure a simulator $\mathsf{SIM}_{\mathsf{OWF}}$ which exhibits the assumption that function $\mathsf{CommittedOwf}[f, \mathsf{COM}]$ (tightly) realizes the OWF functionality $\mathcal{F}_{\mathsf{OWF}}$. Recall that an honest party that evaluates $\mathsf{CommittedOwf}[f, \mathsf{COM}]$ on argument x must output a commitment C_F to $F = f(x)$ to the environment. Consequently, simulator $\mathsf{SIM}_{\mathsf{OWF}}$ must also produce such commitment whenever a real-world party evaluates this function. Since in our context this happens when S initializes the password file, hence SIM engages $\mathsf{SIM}_{\mathsf{OWF}}$ also in the password file initialization.

Game hopping proof. Below we show a sequence of games that bridges between the environment's view of an interaction with the real-world protocol $\mathsf{TFA}_{\mathsf{gen}}$ and the environment's view of an interaction with the UC TFA functionality $\mathcal{F}_{\mathsf{TFA\text{-}KE}}$ which internally interacts with the simulator SIM shown in Figures 11 and 12. To simplify notation we treat the real-world adversary as a "dummy" sub-procedure of the environment $\mathcal{Z}$, hence we assume that $\mathcal{Z}$ interacts directly with all security games. We denote the probability that $\mathcal{Z}$ outputs 1 in the interaction with the real-world game or the simulated game as resp. $\mathbf{Real}_{\mathcal{Z}}[\mathsf{TFA}_{\mathsf{gen}}]$ and $\mathbf{Ideal}_{\mathcal{Z}}[\mathcal{F}_{\mathsf{TFA\text{-}KE}}, \mathsf{SIM}]$. We denote the probability that $\mathcal{Z}$ outputs 1 in an interaction with security game G_i as $\mathsf{Exp}_{\mathcal{Z}}[\mathsf{G}_i] = \Pr[1 \leftarrow \mathcal{Z}^{\mathsf{G}_i}(1^\kappa)]$, and we define $\Delta(\mathsf{G}_i, \mathsf{G}_j) = |\Pr[\mathsf{Exp}_{\mathcal{Z}}[\mathsf{G}_i]] - \Pr[\mathsf{Exp}_{\mathcal{Z}}[\mathsf{G}_j]]|$. The sequence of games below starts with game G_0 s.t. $\mathsf{Exp}_{\mathcal{Z}}[\mathsf{G}_0] = \mathbf{Real}_{\mathcal{Z}}[\mathsf{TFA}_{\mathsf{gen}}]$ and ends with game G_{13} s.t. $\mathsf{Exp}_{\mathcal{Z}}[\mathsf{G}_{13}] = \mathbf{Ideal}_{\mathcal{Z}}[\mathcal{F}_{\mathsf{TFA\text{-}KE}}, \mathsf{SIM}]$, and the theorem follows if we show that $\Delta(\mathsf{G}_0, \mathsf{G}_{13})$ is a negligible function of the security parameter.

Throughout the proof we fix environment $\mathcal{Z}$ and we use q_{sid} for the number of password account initializations $\mathcal{Z}$ creates (via StoreFile calls to some S parties), q_{ses} for the total number of C and S authentication sessions $\mathcal{Z}$ invokes, and q_{H} for the number of $\mathcal{Z}$'s hash oracle calls. Moreover, at some steps we reduce $\mathcal{Z}$'s distinguishing advantage to breaking the binding property of commitment COM, or the hiding property of COM, or the collision resistance of function f^*, or to distinguishing between the real-world evaluation of function f and its *simulation* as a realization of $\mathcal{F}_{\mathsf{OWF}}$ via simulator $\mathsf{SIM}_{\mathsf{OWF}}$. The reductions which convert $\mathcal{Z}$'s distinguishing advantage to an attack against these properties are only a small overlay over $\mathcal{Z}$, so for notational convenience we denote the adversarial advantage of these reductions against these notions as if the corresponding reduction is equated with $\mathcal{Z}$. We thus denote these reduction advantages as resp. $\mathsf{Adv}^{\mathsf{bind}}_{\mathsf{COM}}[\mathcal{Z}]$, $\mathsf{Adv}^{\mathsf{hid}}_{\mathsf{COM}}[\mathcal{Z}]$, $\mathsf{Adv}^{\mathsf{crh}}_{f^*}[\mathcal{Z}]$, and $\mathsf{Adv}^{\mathcal{F}_{\mathsf{OWF}}}_{f,\mathsf{COM}}[\mathcal{Z}, \mathsf{SIM}_{\mathsf{OWF}}]$, all of them functions of the security parameter. (The last advantage stands for a distinguishing advantage of $\mathcal{Z}$ between a real-world view of an honest party evaluating the one-way function $\mathsf{CommittedOwf}[f, \mathsf{COM}]$ and the view created by the simulator $\mathsf{SIM}_{\mathsf{OWF}}$ interacting with the ideal one-way function model $\mathcal{F}_{\mathsf{OWF}}$.)

Game G_0 (real world with inscribed helper functionalities): We start with a game G_0 which models the real-world execution $\mathbf{Real}_{\mathcal{Z}}[\mathsf{TFA}_{\mathsf{gen}}]$, but with inscribed functionalities $\mathcal{F}_{\mathsf{RO}}$, $\mathcal{F}_{\mathsf{NIZK}}$, and $\mathcal{F}_{\mathsf{CKEM}}$. The result is a game, shown in

$\mathsf{Init}^{\mathsf{sid}}(\overline{\mathsf{pw}}[+\pi])$, i.e. ($\textsc{StoreFile}, \mathsf{sid}, \overline{\mathsf{pw}}$) to S and ($\textsc{Proof}$, "init", π) to $\mathcal{F}_{\mathsf{NIZK}}$:

1. $s \leftarrow_{\$} \mathbb{Z}_{2^t}, k \leftarrow_{\$} \{0,1\}^{\kappa}, \overline{\mathsf{hpw}} \leftarrow \mathsf{H}_{\mathsf{p}}(\overline{\mathsf{pw}}), F \leftarrow f^*(\mathsf{sid}, \overline{\mathsf{hpw}}, s)$
2. $(C_F, d_F) \leftarrow \mathsf{COM}(F)$, return C_F to $\mathcal{Z}$ [$\mathcal{Z}$ learns C_F from $\mathcal{F}_{\mathsf{NIZK}}$ interface]
3. receive π from $\mathcal{Z}$ and save $((\mathsf{sid}, C_F), \bot, \pi)$

$\mathsf{O}^{\mathsf{sid}}_{\mathsf{compr}}$, i.e. ($\textsc{CompromiseFile}, \mathsf{sid}$) to S:

– return (k, F, d_F) [values C_F, π are already known]

$\mathsf{H}^{[\mathsf{sid}]}_{\mathsf{name}}(x)$ for $\mathsf{name} \in \{\mathsf{code}, \mathsf{p}\}$, i.e. query to hash functions $\mathsf{H}^{\mathsf{sid}}_{\mathsf{code}}$ and H_{p}:
(same procedure is followed whenever $\mathsf{H}_{\mathsf{code}}$ or H_{p} is called by another oracle)

1. if $\mathsf{H}^{[\mathsf{sid}]}_{\mathsf{name}}(x) \neq \bot$ then return $\mathsf{H}^{[\mathsf{sid}]}_{\mathsf{name}}(x)$ [initially $\mathsf{H}^{[\mathsf{sid}]}_{\mathsf{name}}(x) = \bot$ for all x]
2. else sample y from $\mathsf{H}^{[\mathsf{sid}]}_{\mathsf{name}}$ range and set (and return) $\mathsf{H}^{[\mathsf{sid}]}_{\mathsf{name}}(x) := y$

$\mathsf{D}^{\mathsf{sid}}_{\mathsf{code}}(T)$, i.e. query ($\textsc{GetCode}, \mathsf{sid}, T$) to D:

– return $\mathsf{code} = s - \mathsf{H}^{\mathsf{sid}}_{\mathsf{code}}(k, T)$

$\mathsf{C}_1^{\mathsf{ckid}}(\mathsf{pw}_{\mathsf{C}}, \mathsf{code}, C'_F, \pi')$ for $\mathsf{ckid} = (\mathsf{sid}, \mathsf{ssid})$:
i.e. command ($\textsc{CSession}, ..., \mathsf{pw}_{\mathsf{C}}, \mathsf{code}$) to C and a network message (C'_F, π') to C

– set $\mathsf{hpw}_{\mathsf{C}} = \mathsf{H}_{\mathsf{p}}(\mathsf{pw}_{\mathsf{C}})$, mark $\mathsf{C}^{\mathsf{ckid}}$ as fresh, save $(\mathsf{hpw}_{\mathsf{C}}, \mathsf{code}, C'_F, \pi')$

$\mathsf{S}^{\mathsf{ckid}}(b, T, x)$ for $\mathsf{ckid} = (\mathsf{sid}, \mathsf{ssid})$:
i.e. ($\textsc{SSession}, ..., T$) and ($\textsc{Sout}, \mathsf{ckid}$) [$b = 0$] or ($\textsc{AdvCeval}, \mathsf{ckid}, x$) [$b = 1$]:

1. if $b = 0$ and $\mathsf{C}^{\mathsf{ckid}}$ is not marked fresh then ignore this query
2. if $b = 0$: set $\mathsf{SK}_{\mathsf{S}} \leftarrow \{0,1\}^{\kappa}$, mark $\mathsf{S}^{\mathsf{ckid}}$ as $\mathsf{completed}(T, \mathsf{SK}_{\mathsf{S}})$, return SK_{S}
3. if $b = 1$: set $\mathsf{SK}_{\mathsf{S}} \leftarrow \{0,1\}^{\kappa}$, mark $\mathsf{S}^{\mathsf{ckid}}$ as used, and if $x = (\mathsf{sid}, \mathsf{hpw}, s_{\mathsf{C}})$ s.t. $F = f(\mathsf{sid}, \mathsf{hpw}, s_{\mathsf{C}} + \mathsf{H}^{\mathsf{sid}}_{\mathsf{code}}(k, T))$ then return $(\mathsf{SK}_{\mathsf{S}}, \mathsf{SK}_{\mathsf{S}})$ else return $(\bot, \mathsf{SK}_{\mathsf{S}})$

$\mathsf{C}_2^{\mathsf{ckid}}(b, y, \mathsf{SK}^*, \mathbb{w})$, i.e. ($\textsc{Cout}, \mathsf{ckid}$) [$b = 0$] or ($\textsc{AdvSeval}(\mathsf{ckid}, y, \mathsf{SK}^*)$ [$b = 1$]
followed by ($\textsc{Witness}, \mathbb{w}$) to $\mathcal{F}_{\mathsf{NIZK}}$:

1. if $b = 0$ and $\mathsf{S}^{\mathsf{ckid}}$ is marked used then ignore this query else mark $\mathsf{C}^{\mathsf{ckid}}$ as used
2. if $\exists$ no record $((\mathsf{sid}, C'_F), \cdot, \pi')$ and $\mathcal{Z}$ sends $(\textsc{Witness}, \mathbb{w})$ s.t. $\mathbb{w} = (\mathsf{hpw}', s', {d_F}')$ and $C'_F = \mathsf{COM}(f^*(\mathsf{sid}, \mathsf{hpw}', s'); {d_F}')$ then save $((\mathsf{sid}, C'_F), \mathbb{w}, \pi')$
3. if $\exists$ (still) no record $((\mathsf{sid}, C'_F), \cdot, \pi')$ return $\mathsf{SK}_{\mathsf{C}} \leftarrow_{\$} \{0,1\}^{\kappa}$
4. elseif $b = 0$ and $\mathsf{S}^{\mathsf{ckid}}$ is marked $\mathsf{completed}(T, \mathsf{SK}_{\mathsf{S}})$:
   (a) if $C'_F = C_F$ and $F = f^*(\mathsf{sid}, \mathsf{hpw}_{\mathsf{C}}, \mathsf{code} + \mathsf{H}^{\mathsf{sid}}_{\mathsf{code}}(k, T))$ return $\mathsf{SK}_{\mathsf{C}} := \mathsf{SK}_{\mathsf{S}}$
   (b) else return $\mathsf{SK}_{\mathsf{C}} \leftarrow_{\$} \{0,1\}^{\kappa}$
5. elseif $b = 1$ then:
   (a) if $y = (\mathsf{sid}, F', s_{\mathsf{S}}, {d_F}')$ s.t. $C'_F = \mathsf{COM}(F'; {d_F}')$ and $F' = f^*(\mathsf{sid}, \mathsf{hpw}_{\mathsf{C}}, \mathsf{code} + s_{\mathsf{S}})$ and return $\mathsf{SK}_{\mathsf{C}} = \mathsf{SK}^*$
   (b) else return $\mathsf{SK}_{\mathsf{C}} \leftarrow_{\$} \{0,1\}^{\kappa}$

$\mathsf{O}_{\mathsf{group}}(\cdot)$, i.e. adversary's group operations:

– adversarial group operations are handled by a generic group interface

Fig. 13. Game $\mathbf{Real}_{\mathcal{Z}}[\mathsf{TFA}_{\mathsf{gen}}]$: the real world with inscribed code of $\mathcal{F}_{\mathsf{RO}}, \mathcal{F}_{\mathsf{NIZK}}, \mathcal{F}_{\mathsf{CKEM}}$

Figure 13, where $\mathcal{Z}$ can access oracles $\mathsf{Init}^{\mathsf{sid}}$, $\mathsf{O}^{\mathsf{sid}}_{\mathsf{compr}}$, $\mathsf{D}^{\mathsf{sid}}_{\mathsf{code}}$, $\mathsf{H}^{[\mathsf{sid}]}$, $\mathsf{C}_1^{\mathsf{ckid}}$, $\mathsf{C}_2^{\mathsf{ckid}}$, and $\mathsf{S}^{\mathsf{ckid}}$, corresponding resp. to an initialization call StoreFile to S, a password file compromise call CompromiseFile to S, a code retrieval from the device via call GetCode to D, calls to hash functions $\mathsf{H}_{\mathsf{code}}, \mathsf{H}_{\mathsf{p}}$, and calls to client and server oracles where $\mathsf{C}_1^{\mathsf{ckid}}, \mathsf{C}_2^{\mathsf{ckid}}, \mathsf{S}^{\mathsf{ckid}}$ model resp. the first and second round of C's code and the single round of S. We index these oracles by $\mathsf{ckid} = (\mathsf{sid}, \mathsf{ssid})$ which defines all the relevant identifiers on C and S sessions. The last oracle, $\mathsf{O}_{\mathsf{group}}$, models $\mathcal{Z}$'s operations in a group which is modeled as a generic group in the proof that $\mathsf{CommittedOwf}[f, \mathsf{COM}]$ realizes $\mathcal{F}_{\mathsf{OWF}}$.

The real-world experiment in Figure 13 makes some notational simplifications. When $\mathcal{Z}$ calls Oracles $\mathsf{S}^{\mathsf{ckid}}, \mathsf{C}_1^{\mathsf{ckid}}, \mathsf{C}_2^{\mathsf{ckid}}$, which model S and C sessions, it receives some messages which we omit because they return information which is already known to $\mathcal{Z}$: (1) we omit message (C_F, π) from S which should be triggered by a call to $\mathsf{S}^{\mathsf{ckid}}$, because both values are already known to $\mathcal{Z}$ from an interaction with $\mathsf{Init}^{\mathsf{sid}}$, and for the same reason we combine $\mathcal{Z}$'s call $(\text{SSession}, ...)$ to S with calls $(\text{Sout}, \mathsf{ckid})$ or $(\text{AdvCeval}, \mathsf{ckid}, ...)$ to $\mathcal{F}_{\mathsf{CKEM}}$, since the first one leaks no new information to $\mathcal{Z}$; (2) we omit messages $(\text{Ceval}, \mathsf{ckid}, \mathsf{C}, (\mathsf{sid}, C'_F))$ and $(\text{Verify}, \mathsf{C}, \mathsf{ckid}, (\mathsf{sid}, C'_F), \pi')$ which the adversary would see as messages from resp. $\mathcal{F}_{\mathsf{CKEM}}$ and $\mathcal{F}_{\mathsf{NIZK}}$ in response to a call to $\mathsf{C}_2^{\mathsf{ckid}}$, because all values these messages carry are specified in $\mathcal{Z}$'s call to $\mathsf{C}_1^{\mathsf{ckid}}$. Finally, we note that oracle $\mathsf{S}^{\mathsf{ckid}}$ in Figure 13 return a pair of values, e.g. $(\mathsf{SK}_{\mathsf{S}}, \mathsf{SK}_{\mathsf{S}})$ or $(\perp, \mathsf{SK}_{\mathsf{S}})$, in response to a call $(\text{AdvCeval}, \mathsf{ckid}, ...)$ to $\mathcal{F}_{\mathsf{CKEM}}$, because the first element stands for a value SK returned by the $\mathcal{F}_{\mathsf{CKEM}}$ interface in message $(\text{Cout}, \mathsf{ssid}, \mathsf{SK})$, which can be either $\mathsf{SK} = \mathsf{SK}_{\mathsf{S}}$ or $\mathsf{SK} = \perp$, while the second element stands for the S's output which is passed to the environment. We conclude that:

$$\mathsf{Exp}_{\mathcal{Z}}[\mathsf{G}_0] = \mathbf{Real}_{\mathcal{Z}}[\mathsf{TFA}_{\mathsf{gen}}] \tag{13}$$

Game G_1 (move defining k to the first query to $\mathsf{H}^{\mathsf{sid}}_{\mathsf{code}}$): We modify the game so that key k is not chosen in the call to $\mathsf{Init}^{\mathsf{sid}}$ but in the first query to $\mathsf{H}^{\mathsf{sid}}_{\mathsf{code}}$, and then $\mathsf{Init}^{\mathsf{sid}}$ re-uses this key k. This change will be useful in the next game change but it purely notational, and therefore:

$$\Delta_{\mathcal{Z}}(\mathsf{G}_0, \mathsf{G}_1) = 0 \tag{14}$$

Game G_2 (program $\mathsf{H}^{\mathsf{sid}}_{\mathsf{code}}$ using a random function $\mathsf{R}^{\mathsf{sid}}$): We modify the game so that whenever $\mathsf{H}^{\mathsf{sid}}_{\mathsf{code}}(k, \cdot)$ is called (for k output by the initialization), either directly via $\mathcal{Z}$'s call to $\mathsf{H}^{\mathsf{sid}}_{\mathsf{code}}$ or indirectly by oracles $\mathsf{D}^{\mathsf{sid}}_{\mathsf{code}}$, $\mathsf{S}^{\mathsf{ckid}}$, and $\mathsf{C}_2^{\mathsf{ckid}}$, on an argument for which this value is undefined, the game programs its value using a random function $\mathsf{R}^{\mathsf{sid}} : \{0,1\}^* \to \{0,1\}^t$. Specifically, we set $\mathsf{H}^{\mathsf{sid}}_{\mathsf{code}}(k, T) = s - \mathsf{R}^{\mathsf{sid}}(T)$ for all T, where k is the unique PRF key chosen for this sid by $\mathsf{Init}^{\mathsf{sid}}$. (Note that by the rules of G_1 oracle $\mathsf{H}^{\mathsf{sid}}_{\mathsf{code}}$ can act differently on input $(k, \cdot)$ for k which will be output by the initialization even before $\mathcal{Z}$ calls $\mathsf{Init}^{\mathsf{sid}}$.) Since both $\mathsf{H}^{\mathsf{sid}}_{\mathsf{code}}(k, \cdot)$ and $\mathsf{R}^{\mathsf{sid}}(\cdot)$ are random functions on the same range, this is an internal game re-write which makes no difference to an entity

that interacts with it, hence:

$$\Delta_{\mathcal{Z}}(\mathsf{G}_1, \mathsf{G}_2) = 0 \tag{15}$$

Game G_3 (using function $\mathsf{R}^{\mathsf{sid}}$ instead of $\mathsf{H}^{\mathsf{sid}}_{\mathsf{code}}$, defining s indirectly): We make the following modifications to G_2:

1. Oracle $\mathsf{Init}^{\mathsf{sid}}$ doesn't choose s, and instead chooses $v_1 \leftarrow_{\$} \mathbb{Z}_{2^t}$, and sets $F \leftarrow f^*(\mathsf{sid}, \overline{\mathsf{hpw}}, v_1 + \mathsf{R}^{\mathsf{sid}}(1))$, which implicitly defines s as $s = v_1 + \mathsf{R}^{\mathsf{sid}}(1)$.
2. Oracle $\mathsf{H}^{\mathsf{sid}}_{\mathsf{code}}$ on $x = (k, T)$ for any T, returns $(v_1 + \mathsf{R}^{\mathsf{sid}}(1)) - \mathsf{R}^{\mathsf{sid}}(T)$.
3. Oracle $\mathsf{D}^{\mathsf{sid}}_{\mathsf{code}}(T)$ returns $\mathsf{R}^{\mathsf{sid}}(T)$.
4. Oracle $\mathsf{S}^{\mathsf{ckid}}$ for $b = 1$ changes condition $F = f^*(\mathsf{sid}, \mathsf{hpw}, s_{\mathsf{C}} + \mathsf{H}^{\mathsf{sid}}_{\mathsf{code}}(k, T))$ to condition $F = f^*(\mathsf{sid}, \mathsf{hpw}, (v_1 + \mathsf{R}^{\mathsf{sid}}(1)) + (s_{\mathsf{C}} - \mathsf{R}^{\mathsf{sid}}(T)))$.
5. Oracle $\mathsf{C}^{\mathsf{ckid}}_2$ for $b = 0$ changes condition $F = f^*(\mathsf{sid}, \mathsf{hpw}_{\mathsf{C}}, \mathsf{code} + \mathsf{H}^{\mathsf{sid}}_{\mathsf{code}}(k, T))$ to condition $F = f^*(\mathsf{sid}, \mathsf{hpw}_{\mathsf{C}}, (v_1 + \mathsf{R}^{\mathsf{sid}}(1)) + (\mathsf{code} - \mathsf{R}^{\mathsf{sid}}(T)))$

The first change is only a renaming of variables: Instead of choosing s in $\mathbb{Z}_{2^t}$ directly, we sample random v_1 in $\mathbb{Z}_{2^t}$ and define s as $s = v_1 + \mathsf{R}^{\mathsf{sid}}(1)$. The second change follows because $\mathsf{H}^{\mathsf{sid}}_{\mathsf{code}}(k, T) = s - \mathsf{R}^{\mathsf{sid}}(T) = (v_1 + \mathsf{R}^{\mathsf{sid}}(1)) - \mathsf{R}^{\mathsf{sid}}(T)$. The third change follows because if $\mathsf{H}^{\mathsf{sid}}_{\mathsf{code}}(k, T) = s - \mathsf{R}^{\mathsf{sid}}(T)$ then $\mathsf{D}^{\mathsf{sid}}_{\mathsf{code}}(T) = s - \mathsf{H}^{\mathsf{sid}}_{\mathsf{code}}(k, T) = s - (s - \mathsf{R}^{\mathsf{sid}}(T)) = \mathsf{R}^{\mathsf{sid}}(T)$. The fourth change follows because if $\mathsf{H}^{\mathsf{sid}}_{\mathsf{code}}(k, T) = s - \mathsf{R}^{\mathsf{sid}}(T)$ and $s = v_1 + \mathsf{R}^{\mathsf{sid}}(1)$ then $s_{\mathsf{C}} + \mathsf{H}^{\mathsf{sid}}_{\mathsf{code}}(k, T) = s_{\mathsf{C}} + (s - \mathsf{R}^{\mathsf{sid}}(T)) = (v_1 + \mathsf{R}^{\mathsf{sid}}(1)) + (s_{\mathsf{C}} - \mathsf{R}^{\mathsf{sid}}(T))$. Replacing s_{C} with code in these equations also shows that the fifth change is also only a re-write. Since all these changes are only notational we conclude:

$$\Delta_{\mathcal{Z}}(\mathsf{G}_2, \mathsf{G}_3) = 0 \tag{16}$$

Game G_4 (postpone choice of k to file compromise): We modify the game so that (1) key k is not picked by the first call to $\mathsf{H}^{\mathsf{sid}}_{\mathsf{code}}$ but at $\mathsf{O}^{\mathsf{sid}}_{\mathsf{compr}}$, (2) oracle $\mathsf{H}^{\mathsf{sid}}_{\mathsf{code}}$ does not program $\mathsf{H}^{\mathsf{sid}}_{\mathsf{code}}$ outputs on $x = (k, \cdot)$ before password file compromise (because k is not yet defined then), but performs this programming on $(k, \cdot)$ queries after $\mathsf{O}^{\mathsf{sid}}_{\mathsf{compr}}$ picks and reveals k, and (3) we introduce an abort, denoted $[\mathit{abort.key}]$ to $\mathsf{O}^{\mathsf{sid}}_{\mathsf{compr}}$ if it picks $k \leftarrow_{\$} \{0,1\}^{\kappa}$ s.t. $\mathcal{Z}$ made a prior query of the form $x = (k, \cdot)$, i.e. if $\mathsf{H}^{\mathsf{sid}}_{\mathsf{code}}(k, T) \neq \perp$ for any T.

Game G_3 and G_4 are identical conditioned on the event that $\mathcal{Z}$ does not query $\mathsf{O}^{\mathsf{sid}}_{\mathsf{compr}}$ on an argument of the form $x = (k, \cdot)$ before $\mathcal{Z}$ queries $\mathsf{O}^{\mathsf{sid}}_{\mathsf{compr}}$. Indeed, in G_3 the only oracle which uses k apart of $\mathsf{O}^{\mathsf{sid}}_{\mathsf{compr}}$ is $\mathsf{H}^{\mathsf{sid}}_{\mathsf{code}}$, so if $\mathcal{Z}$ doesn't make a query of the form $(k, \cdot)$ to $\mathsf{H}^{\mathsf{sid}}_{\mathsf{code}}$ before it queries $\mathsf{O}^{\mathsf{sid}}_{\mathsf{compr}}$ then the two games are identical. Since k is randomly sampled in $\{0,1\}^{\kappa}$, the probability that $\mathcal{Z}$ does make such query to $\mathsf{H}^{\mathsf{sid}}_{\mathsf{code}}$ before a query to $\mathsf{O}^{\mathsf{sid}}_{\mathsf{compr}}$ is bounded by $q_{\mathsf{H}}/2^{\kappa}$. Since the full bound is that this does not happen for any sid, we conclude:

$$\Delta_{\mathcal{Z}}(\mathsf{G}_3, \mathsf{G}_4) \leq (q_{\mathsf{sid}} \cdot q_{\mathsf{H}})/2^{\kappa} \tag{17}$$

Game G_5 (abort on attack on C for $C'_F = C_F$ but $F' \neq F$): We modify the game to introduce an abort, denoted $[\mathit{abort.cmt1}]$, in the processing of oracle

$\mathsf{C}_2^{\mathsf{ckid}}$ in case $b = 1$, i.e. an active attack on C's CKEM session, if it holds that $C'_F = C_F$, and $y = (\mathsf{sid}, F', s_\mathsf{S}, {d_F}')$ s.t. $C'_F = \mathsf{COM}(F'; {d_F}')$, but $F' \neq F$. The difference between G_4 and G_5 is upper-bounded by the probability that the above condition holds. If we denote the event that this happens on the i-th call to $\mathsf{C}_2^{\mathsf{ckid}}$ by $\mathsf{E_i}$, then $\Pr_{\mathcal{Z}}[\mathsf{E_i}]$ is upper-bounded by $\mathsf{Adv}^{\mathsf{COM}}_{\mathsf{bind}}[\mathcal{Z}]$, because an environment $\mathcal{Z}$ which causes event $\mathsf{E_i}$ allows for an immediate reduction against the binding property of commitment COM. Since the difference between G_4 and G_5 is upper-bounded by $\Pr[\cup_{i=1}^{q_{\mathsf{ses}}} E_i]$ we conclude:

$$\Delta_{\mathcal{Z}}(\mathsf{G}_4, \mathsf{G}_5) \leq q_{\mathsf{ses}} \cdot \mathsf{Adv}^{\mathsf{bind}}_{\mathsf{COM}}[\mathcal{Z}] \tag{18}$$

Game G_6 (abort on attack on C for $C'_F = C_F$ before file compromise): We modify the game to introduce an abort, denoted [*abort.cmt2*], in the processing of oracle $\mathsf{C}_2^{\mathsf{ckid}}$ in the same case $b = 1$ as above, if it holds that $C'_F = C_F$, and $y = (\mathsf{sid}, F', s_\mathsf{S}, {d_F}')$ s.t. $C'_F = \mathsf{COM}(F'; {d_F}')$, and the file compromise oracle $\mathsf{O}^{\mathsf{sid}}_{\mathsf{compr}}$ has not been queried yet.

Note that the above conditions imply that on some call to $\mathsf{C}_2^{\mathsf{ckid}}$, $\mathcal{Z}$ provides $(F', {d_F}')$ s.t. $C_F = \mathsf{COM}(F'; {d_F}')$. Moreover, by G_5 this introduces a difference only in the case that $F' = F$. Let's denote the event that this happens on i-th call to $\mathsf{C}_2^{\mathsf{ckid}}$ by $\mathsf{E_i}$, and let $\epsilon = \Pr_{\mathcal{Z}}[\mathsf{E_i}]$. Extending $\mathcal{Z}$ by the code of $\mathsf{Init}^{\mathsf{sid}}$ which computes F, note that $\mathcal{Z}$ receives $C_F \leftarrow_\$ \mathsf{COM}(F)$, and with probability ϵ creates ${d_F}'$ s.t. $C_F = \mathsf{COM}(F; {d_F}')$. Such $\mathcal{Z}$ implies a reduction against the hiding property of this commitment: The reduction runs $\mathcal{Z}$ and outputs two messages, $m_0 = F$ and $m_1 \neq F$, gets a challenge $C^* \leftarrow_\$ \mathsf{COM}(m_b)$, and sends it to $\mathcal{Z}$ in response to $\mathsf{Init}^{\mathsf{sid}}$, and if $\mathcal{Z}$ outputs d on the i-th $\mathsf{C}_2^{\mathsf{ckid}}$ session s.t. $C^* = \mathsf{COM}(F; d)$ the reduction outputs 1. Since for $b = 0$ this probability is ϵ, and for $b = 1$ the probability that $C^* = \mathsf{COM}(F; d)$ is upper-bounded by $\mathsf{Adv}^{\mathsf{bind}}_{\mathsf{COM}}[\mathcal{Z}]$, we conclude that $\epsilon \leq \mathsf{Adv}^{\mathsf{hid}}_{\mathsf{COM}}[\mathcal{Z}] + \mathsf{Adv}^{\mathsf{bind}}_{\mathsf{COM}}[\mathcal{Z}]$. Since the difference between G_5 and G_6 is upper-bounded by $\Pr[\cup_{i=1}^{q_{\mathsf{ses}}} E_i]$ we conclude:

$$\Delta_{\mathcal{Z}}(\mathsf{G}_5, \mathsf{G}_6) \leq q_{\mathsf{ses}} \cdot (\mathsf{Adv}^{\mathsf{hid}}_{\mathsf{COM}}[\mathcal{Z}] + \mathsf{Adv}^{\mathsf{bind}}_{\mathsf{COM}}[\mathcal{Z}]) \tag{19}$$

Game G_7 (using f^* inputs in equality checks): We modify the game so that whenever $\mathsf{S}^{\mathsf{ckid}}$ or $\mathsf{C}_2^{\mathsf{ckid}}$ performs a check of the form $Y \equiv f^*(x')$ for some Y for which the game knows x s.t. $Y = f^*(x)$, we replace each such check by the check that $x' = x$. Specifically:

1. In $\mathsf{S}^{\mathsf{ckid}}$, in case $b = 1$, game G_7 replaces the check if $F = f^*(\mathsf{sid}, \mathsf{hpw}, (v_1 + \mathsf{R}^{\mathsf{sid}}(1)) + (s_\mathsf{C} - \mathsf{R}^{\mathsf{sid}}(T)))$ with a check if $\overline{\mathsf{hpw}} = \mathsf{hpw}$ and $s_\mathsf{C} = \mathsf{R}^{\mathsf{sid}}(T)$.
2. In $\mathsf{C}_2^{\mathsf{ckid}}$, in case $b = 0$, game G_7 replaces the check if $F = f^*(\mathsf{sid}, \mathsf{hpw}_\mathsf{C}, (v_1 + \mathsf{R}^{\mathsf{sid}}(1)) + (\mathsf{code} - \mathsf{R}^{\mathsf{sid}}(T)))$ with a check if $\overline{\mathsf{hpw}} = \mathsf{hpw}_\mathsf{C}$ and $\mathsf{code} = \mathsf{R}^{\mathsf{sid}}(T)$.
3. In $\mathsf{C}_2^{\mathsf{ckid}}$, in case $b = 1$ and $C'_F = C_F$, and there is no abort, which by $\mathsf{G}_5, \mathsf{G}_6$ implies that $F' = F$ and the password file is compromised, game G_7 replaces the check if $F = f^*(\mathsf{sid}, \mathsf{hpw}_\mathsf{C}, \mathsf{code} + s_\mathsf{S})$ with a check if $\overline{\mathsf{hpw}} = \mathsf{hpw}_\mathsf{C}$ and $\mathsf{code} + s_\mathsf{S} = v_1 + \mathsf{R}^{\mathsf{sid}}(1)$.

4. In $\mathsf{C}_2^{\mathsf{ckid}}$, in case $b = 1$ and $C'_F \neq C_F$, game G_7 replaces the check if $F' = f^*(\mathsf{sid}, \mathsf{hpw}_\mathsf{C}, \mathsf{code} + s_\mathsf{S})$ with a check if $\mathsf{hpw}' = \mathsf{hpw}_\mathsf{C}$ and $s' = \mathsf{code} + s_\mathsf{S}$, for hpw', s' retrieved from the witness $\mathbb{w} = (\mathsf{sid}, \mathsf{hpw}', s')$.

The first three changes are based on the fact that in G_6 $F = f^*(\mathsf{sid}, \overline{\mathsf{hpw}}, v_1 + R(1))$, and the last change is based on the fact that if G_6 admits $\mathbb{w}$ sent by $\mathcal{Z}$ in a query to $\mathsf{C}_2^{\mathsf{ckid}}$ oracle then $F' = f^*(\mathsf{sid}, \mathsf{hpw}', s')$.

The only divergence G_7 creates from G_6 is in the case where some condition of the form $F = f^*(x')$ holds for $F = f^*(x)$, but it does not hold that $x = x'$. However, if $\mathcal{Z}$ ever triggers such case then it also breaks the collision-resistance property of function f^* for the given sid. Since G_6 and G_7 are identical unless $\mathcal{Z}$ finds a collision in f^*, we conclude:

$$\Delta_{\mathcal{Z}}(\mathsf{G}_6, \mathsf{G}_7) \leq \mathsf{Adv}^{\mathsf{crh}}_{f^*}[\mathcal{Z}] \tag{20}$$

Game G_8 (abort on H_p collision): We modify the game by introducing an abort in H_p processing, denoted [*abort.hash*], which is triggered when there occurs a collision in hash H_p, i.e. if for a new argument pw and a fresh sampled value $\mathsf{H}_\mathsf{p}(\mathsf{pw})$ there exists a prior query $\mathsf{pw}' \neq \mathsf{pw}$ s.t. $\mathsf{H}_\mathsf{p}(\mathsf{pw}) = \mathsf{H}_\mathsf{p}(\mathsf{pw}')$. Since H_p outputs are sampled in range $\mathcal{HP}$, by the birthday bound we have:

$$\Delta_{\mathcal{Z}}(\mathsf{G}_7, \mathsf{G}_8) \leq (q_\mathsf{H})^2/|\mathcal{HP}| \tag{21}$$

Game G_9 (using H_p inputs in tests): We modify the game as follows:

1. In $\mathsf{C}_2^{\mathsf{ckid}}$, if $b = 0$ or if ($b = 1$, $C'_F = C_F$, and the password file is compromised), we replace the check $\overline{\mathsf{hpw}} = \mathsf{hpw}_\mathsf{C}$ (see G_7, case 2 and 3) with the check that $\overline{\mathsf{pw}} = \mathsf{pw}_\mathsf{C}$.
2. In $\mathsf{C}_2^{\mathsf{ckid}}$, if $b = 1$ and $C'_F \neq C_F$, we replace the check $\mathsf{hpw}' = \mathsf{hpw}_\mathsf{C}$ (see G_7, case 4), with the check that $\mathcal{Z}$ queried H_p on pw_C and $\mathsf{hpw}' = \mathsf{H}_\mathsf{p}(\mathsf{pw}_\mathsf{C})$.
3. Oracle $\mathsf{C}_1^{\mathsf{ckid}}$ skips computing hpw_C and saves pw_C instead.
4. Before server compromise, oracle $\mathsf{S}^{\mathsf{ckid}}$ replaces check that $\mathsf{hpw} = \overline{\mathsf{hpw}}$ (see G_7, case 1), with the check that $\mathcal{Z}$ queried H_p on $\overline{\mathsf{pw}}$ and $\mathsf{hpw} = \mathsf{H}_\mathsf{p}(\overline{\mathsf{pw}})$.

By the changes in G_8, the first modification does not change the external behavior of the game, because password hash outputs can match only if the inputs match, and otherwise game G_8 aborts. Regarding the second modification, if the environment does not compute hpw_C as $\mathsf{H}_\mathsf{p}(\mathsf{pw}_\mathsf{C})$ then the only information the environment sees about hpw_C is via the $\mathcal{F}_{\mathsf{CKEM}}$ interface, which allows only for testing whether a provided guess is equal to hpw_C. Therefore, distinguishing this modification requires guessing hpw_C on some session without computing it as $\mathsf{H}_\mathsf{p}(\mathsf{pw}_\mathsf{C})$, which happens with probability at most $q_{\mathsf{ses}}/|\mathcal{HP}|$. The third modification is not externally observable once the first two modifications take place. Finally, until password file compromise the only information about $\overline{\mathsf{hpw}}$ which the environment sees without computing $\mathsf{H}_\mathsf{p}(\overline{\mathsf{pw}})$ is in commitment C_F, and via the $\mathcal{F}_{\mathsf{CKEM}}$ interface which allows for testing any guessed value. By commitment hiding, if the probability that the environment guesses $\overline{\mathsf{hpw}}$ given

the commitment C_F to a value which is a function of $\overline{\mathsf{hpw}}$ is ϵ, then the same guess probability without this commitment must be at least $\epsilon - \mathsf{Adv}^{\mathsf{hid}}_{\mathsf{COM}}[\mathcal{Z}]$, but since the latter probability is upper-bounded by $q_{\mathsf{ses}}/|\mathcal{HP}|$ it follows that $\epsilon \leq q_{\mathsf{ses}}/|\mathcal{HP}| + \mathsf{Adv}^{\mathsf{hid}}_{\mathsf{COM}}[\mathcal{Z}]$. The total distinguishing difference is therefore:

$$\Delta_{\mathcal{Z}}(\mathsf{G}_8, \mathsf{G}_9) \leq 2 \cdot q_{\mathsf{H}}/|\mathcal{HP}| + \mathsf{Adv}^{\mathsf{hid}}_{\mathsf{COM}}[\mathcal{Z}] \tag{22}$$

Game G_{10} (using OWF simulator): Game G_9 looks very similar to the ideal-world game except for the following aspects:

1. At initialization query, $\mathsf{Init}^{\mathsf{sid}}$ picks random v_1 and evaluates $\mathsf{CommittedOwf}[f, \mathsf{COM}]$ on input $x = (\mathsf{sid}, \overline{\mathsf{pw}}, v_1 + \mathsf{R}^{\mathsf{sid}}(1))$ as $F = f(x)$, which involves computing $\overline{\mathsf{hpw}} = \mathsf{H}_{\mathsf{p}}(\overline{\mathsf{pw}})$, and $(C_F, d_F) \leftarrow_{\$} \mathsf{COM}(F)$, and reveals C_F.
2. On file compromise query, $\mathsf{O}^{\mathsf{sid}}_{\mathsf{compr}}$ reveals F, d_F created in the initialization.
3. Server oracle $\mathsf{S}^{\mathsf{ckid}}$, after file compromise, in response to an active attack, i.e. in the case $b = 1$, outputs $(\mathsf{SK}_{\mathsf{S}}, \mathsf{SK}_{\mathsf{S}})$ rather than $(\mathsf{SK}_{\perp}, \mathsf{SK}_{\mathsf{S}})$ if $\mathcal{Z}$'s input $x = (\mathsf{sid}, \mathsf{hpw}, s_{\mathsf{C}})$ satisfies that $\mathsf{hpw} = \overline{\mathsf{hpw}}$ and $s_{\mathsf{C}} = \mathsf{R}^{\mathsf{sid}}(1)$.
4. Group operation oracle $\mathsf{O}_{\mathsf{group}}$ is handled by a generic group interface.

The ideal-world game, shown in Figure 14, matches the view of G_9 except for the aspects listed above, which it handles as follows, using simulator $\mathsf{SIM}_{\mathsf{OWF}}$ implied by the assumption that function $\mathsf{CommittedOwf}[f, \mathsf{COM}]$ realizes OWF functionality $\mathcal{F}_{\mathsf{OWF}}$:

1. At initialization query, $\mathsf{Init}^{\mathsf{sid}}$ runs the simulator $\mathsf{SIM}_{\mathsf{OWF}}$ on query $(\textsc{Eval}, \mathsf{sid}, \mathsf{S})$, and receives C_F output by $\mathsf{SIM}_{\mathsf{OWF}}$.
2. On file compromise query, $\mathsf{O}^{\mathsf{sid}}_{\mathsf{compr}}$ acts as the environment to $\mathsf{SIM}_{\mathsf{OWF}}$ and issues to it a file compromise permission, which triggers $\mathsf{SIM}_{\mathsf{OWF}}$ to output $(\textsc{LeakVal}, \mathsf{sid})$. When in response $\mathsf{SIM}_{\mathsf{OWF}}$ outputs F, d_F, the game passes these to the environment and picks random v_1. Importantly, in-between sending $(\textsc{LeakVal}, \mathsf{sid})$ and F, d_F simulator $\mathsf{SIM}_{\mathsf{OWF}}$ might issue multiple $(\textsc{AdvTest}, \mathsf{sid}, ...)$ queries which the game services as described in item 4 below.
3. Server oracle $\mathsf{S}^{\mathsf{ckid}}$, in the same case as item 3, replaces check $\mathsf{hpw} = \overline{\mathsf{hpw}}$ with a check that $\mathcal{Z}$ computed H_{p} on $\overline{\mathsf{pw}}$ and $\mathsf{hpw} = \mathsf{H}_{\mathsf{p}}(\overline{\mathsf{pw}})$.
4. Group operation oracle is handled by $\mathsf{SIM}_{\mathsf{OWF}}$ who monitors adversarial group operations, and after $\mathcal{Z}$'s query to $\mathsf{O}^{\mathsf{sid}}_{\mathsf{compr}}$ which prompts $\mathsf{SIM}_{\mathsf{OWF}}$ to issue $(\textsc{LeakVal}, \mathsf{sid})$, for every query $(\textsc{AdvTest}, \mathsf{sid}, \mathsf{pw}^*, s^*)$ which $\mathsf{SIM}_{\mathsf{OWF}}$ makes, the game sends to $\mathsf{SIM}_{\mathsf{OWF}}$ a response "correct" if $(\mathsf{pw}^*, s^* - v_1) = (\overline{\mathsf{pw}}, \mathsf{R}^{\mathsf{sid}}(1))$ and "wrong" otherwise.

$$\Delta_{\mathcal{Z}}(\mathsf{G}_9, \mathsf{G}_{10}) \leq \mathsf{Adv}^{\mathcal{F}_{\mathsf{OWF}}}_{f,\mathsf{COM}}[\mathcal{Z}, \mathsf{SIM}_{\mathsf{OWF}}] \tag{23}$$

Game G_{11} (postpone choice of v_1 to file compromise): We modify the game so v_1 is sampled not at the initialization call to $\mathsf{Init}^{\mathsf{sid}}$ but on the file compromise call to $\mathsf{O}^{\mathsf{sid}}_{\mathsf{compr}}$. Since G_{10} doesn't use v_1 until file compromise we have:

$$\Delta_{\mathcal{Z}}(\mathsf{G}_{10}, \mathsf{G}_{11}) = 0 \tag{24}$$

$\mathsf{Init}^{\mathsf{sid}}(\overline{\mathsf{pw}}[+\pi])$, i.e. ($\textsc{StoreFile}, \mathsf{sid}, \overline{\mathsf{pw}}$) to S and ($\textsc{Proof}$, "init", π) to $\mathcal{F}_{\mathsf{NIZK}}$:

1. set C_F to output of $\mathsf{SIM}_{\mathsf{OWF}}$ on ($\textsc{Eval}, \mathsf{sid}, \mathsf{S}$)
2. return C_F to $\mathcal{Z}$ [$\mathcal{Z}$ learns C_F from $\mathcal{F}_{\mathsf{NIZK}}$ interface]
3. receive π from $\mathcal{Z}$ and save $((\mathsf{sid}, C_F), \perp, \pi)$

$\mathsf{O}^{\mathsf{sid}}_{\mathsf{compr}}$, i.e. ($\textsc{CompromiseFile}, \mathsf{sid}$) to S:

1. issue permission to $\mathsf{SIM}_{\mathsf{OWF}}$ to output ($\textsc{LeakVal}, \mathsf{sid}$)
2. (service $\mathsf{SIM}_{\mathsf{OWF}}$ queries generated by $\mathsf{O}_{\mathsf{group}}$ oracle as below)
3. wait for $\mathsf{SIM}_{\mathsf{OWF}}$ to output (F, d_F)
4. set $v_1 \leftarrow_{\$} \mathbb{Z}_{2^t}, k \leftarrow_{\$} \{0,1\}^{\kappa}$, mark sid as $\mathsf{compromised}$
5. abort [*abort.key*] if $\mathsf{H}^{\mathsf{sid}}_{\mathsf{code}}(k, T)$ has been queried before for any T
6. return (k, F, d_F) [values C_F, π are already known]

$\mathsf{H}^{[\mathsf{sid}]}_{\mathsf{name}}(x)$ for $\mathsf{name} \in \{\mathsf{code}, \mathsf{p}\}$, i.e. query to hash functions $\mathsf{H}^{\mathsf{sid}}_{\mathsf{code}}$ and H_{p}:

1. if $\mathsf{H}^{[\mathsf{sid}]}_{\mathsf{name}}(x) \neq \perp$ then return $\mathsf{H}^{[\mathsf{sid}]}_{\mathsf{name}}(x)$ [initially $\mathsf{H}^{[\mathsf{sid}]}_{\mathsf{name}}(x) = \perp$ for all x]
2. elseif $\mathsf{name} = \mathsf{code}$, sid is $\mathsf{compromised}$, and $x = (k, T)$ for some T, then set (and return) $\mathsf{H}^{\mathsf{sid}}_{\mathsf{code}}(k, T) := v_1 + \mathsf{R}^{\mathsf{sid}}(1) - \mathsf{R}^{\mathsf{sid}}(T)$. [$\mathsf{R}^{\mathsf{sid}}$ is a random function]
3. else sample y from $\mathsf{H}^{[\mathsf{sid}]}_{\mathsf{name}}$ range and set (and return) $\mathsf{H}^{[\mathsf{sid}]}_{\mathsf{name}}(x) := y$, but if $\mathsf{name} = \mathsf{p}$ and H_{p} encounters a collision then abort [*abort.hash*]

$\mathsf{D}^{\mathsf{sid}}_{\mathsf{code}}(T)$, i.e. query ($\textsc{GetCode}, \mathsf{sid}, T$) to D:

– return $\mathsf{R}^{\mathsf{sid}}(T)$

$\mathsf{C}^{\mathsf{ckid}}_1(\mathsf{pw}_{\mathsf{C}}, \mathsf{code}, C'_F, \pi')$ for $\mathsf{ckid} = (\mathsf{sid}, \mathsf{ssid})$:
i.e. command ($\textsc{CSession}, ..., \mathsf{pw}_{\mathsf{C}}, \mathsf{code}$) to C and a network message (C'_F, π') to C

– save $(\mathsf{pw}_{\mathsf{C}}, \mathsf{code}, C'_F, \pi')$, mark $\mathsf{C}^{\mathsf{ckid}}$ as fresh

$\mathsf{S}^{\mathsf{ckid}}(b, T, x)$ for $\mathsf{ckid} = (\mathsf{sid}, \mathsf{ssid})$:
i.e. ($\textsc{SSession}, ..., T$) and ($\textsc{Sout}, \mathsf{ckid}$) [$b = 0$] or ($\textsc{AdvCeval}, \mathsf{ckid}, x$) [$b = 1$]:

1. if $b = 0$ and $\mathsf{C}^{\mathsf{ckid}}$ is not marked fresh then ignore this query
2. if $b = 0$: set $\mathsf{SK}_{\mathsf{S}} \leftarrow \{0,1\}^{\kappa}$, mark $\mathsf{S}^{\mathsf{ckid}}$ as $\mathsf{completed}(T, \mathsf{SK}_{\mathsf{S}})$, return SK_{S}
3. if $b = 1$: set $\mathsf{SK}_{\mathsf{S}} \leftarrow \{0,1\}^{\kappa}$, mark $\mathsf{S}^{\mathsf{ckid}}$ as used, and if $x = (\mathsf{sid}, \mathsf{hpw}, s_{\mathsf{C}})$ s.t. $\mathsf{hpw} = \mathsf{H}_{\mathsf{p}}(\overline{\mathsf{pw}})$ and $s_{\mathsf{C}} = \mathsf{R}^{\mathsf{sid}}(T)$ then return $(\mathsf{SK}_{\mathsf{S}}, \mathsf{SK}_{\mathsf{S}})$ else return $(\perp, \mathsf{SK}_{\mathsf{S}})$

$\mathsf{C}^{\mathsf{ckid}}_2(b, y, \mathsf{SK}^*, \mathbb{w})$, i.e. ($\textsc{Cout}, \mathsf{ckid}$) [$b = 0$] or ($\textsc{AdvSeval}(\mathsf{ckid}, y, \mathsf{SK}^*)$ [$b = 1$] followed by ($\textsc{Witness}, \mathbb{w}$) to $\mathcal{F}_{\mathsf{NIZK}}$:

1. if $b = 0$ and $\mathsf{S}^{\mathsf{ckid}}$ is marked used then ignore this query else mark $\mathsf{C}^{\mathsf{ckid}}$ as used
2. if $\exists$ no record $((\mathsf{sid}, C'_F), \cdot, \pi')$ and $\mathcal{Z}$ sends ($\textsc{Witness}, \mathbb{w}$) s.t. $\mathbb{w} = (\mathsf{hpw}, s, {d_F}')$ and $C'_F = \mathsf{COM}(f^*(\mathsf{sid}, \mathsf{hpw}, s); {d_F}')$ then save $((\mathsf{sid}, C'_F), \mathbb{w}, \pi')$
3. if $\exists$ (still) no record $((\mathsf{sid}, C'_F), \cdot, \pi')$ return $\mathsf{SK}_{\mathsf{C}} \leftarrow_{\$} \{0,1\}^{\kappa}$
4. elseif $b = 0$ and $\mathsf{S}^{\mathsf{ckid}}$ is marked $\mathsf{completed}(T, \mathsf{SK}_{\mathsf{S}})$:
   (a) if $C'_F = C_F$ and $(\mathsf{pw}_{\mathsf{C}}, \mathsf{code}) = (\overline{\mathsf{pw}}, \mathsf{R}^{\mathsf{sid}}(T))$ return $\mathsf{SK}_{\mathsf{C}} := \mathsf{SK}_{\mathsf{S}}$
   (b) else return $\mathsf{SK}_{\mathsf{C}} \leftarrow_{\$} \{0,1\}^{\kappa}$
5. elseif $b = 1$ then:
   (a) if $y \neq (\mathsf{sid}, F', s_{\mathsf{S}}, {d_F}')$ s.t. $C'_F = \mathsf{COM}(F'; {d_F}')$ return $\mathsf{SK}_{\mathsf{C}} \leftarrow_{\$} \{0,1\}^{\kappa}$
   (b) elseif $C'_F = C_F$ then:
      i. if sid is not $\mathsf{compromised}$ then abort [*abort.cmt*]
      ii. elseif $(\mathsf{pw}_{\mathsf{C}}, \mathsf{code}) = (\overline{\mathsf{pw}}, \mathsf{R}^{\mathsf{sid}}(1) + v_1 - s_{\mathsf{S}})$ then return $\mathsf{SK}_{\mathsf{C}} := \mathsf{SK}^*$
      iii. else return $\mathsf{SK}_{\mathsf{C}} \leftarrow_{\$} \{0,1\}^{\kappa}$
   (c) else retrieve $((\mathsf{sid}, C'_F), \mathbb{w}, \pi')$ s.t. $\mathbb{w} = (\mathsf{hpw}', s', {d_F}')$, and:
      i. if $\mathsf{hpw}' = \mathsf{H}_{\mathsf{p}}(\mathsf{pw}_{\mathsf{C}})$ and $\mathsf{code} = s' - s_{\mathsf{S}}$ return $\mathsf{SK}_{\mathsf{C}} := \mathsf{SK}^*$
      ii. else return $\mathsf{SK}_{\mathsf{C}} \leftarrow_{\$} \{0,1\}^{\kappa}$

$\mathsf{O}_{\mathsf{group}}(\cdot)$, i.e. adversary's group operations:

– $\mathsf{SIM}_{\mathsf{OWF}}$ monitors adversarial group operations
– after $\mathcal{Z}$'s query to $\mathsf{O}^{\mathsf{sid}}_{\mathsf{compr}}$, which prompts $\mathsf{SIM}_{\mathsf{OWF}}$ to issue ($\textsc{LeakVal}, \mathsf{sid}$), for every query ($\textsc{AdvTest}, \mathsf{sid}, \mathsf{pw}^*, s^*$) which $\mathsf{SIM}_{\mathsf{OWF}}$ makes (as if to $\mathcal{F}_{\mathsf{OWF}}$), it receives "correct" if $(\mathsf{pw}^*, s^* - v_1) = (\overline{\mathsf{pw}}, \mathsf{R}^{\mathsf{sid}}(1))$ and "wrong" otherwise.

Fig. 14. Experiment $\mathbf{Ideal}_{\mathcal{Z}}[\mathcal{F}_{\mathsf{TFA\text{-}KE}}, \mathsf{SIM}]$: the ideal world implied by $\mathcal{F}_{\mathsf{TFA\text{-}KE}}$ and SIM

Game G_{12} (use H_p pre-image in server oracle test): We modify the game so $\mathsf{S}^{\mathsf{ckid}}$ oracle in the active attack case, i.e. $b = 1$, after server compromise, replaces check $\mathsf{hpw} = \overline{\mathsf{hpw}}$ with a check that $\mathcal{Z}$ computed H_p on $\overline{\mathsf{pw}}$ and $\mathsf{hpw} = \mathsf{H}_\mathsf{p}(\overline{\mathsf{pw}})$. Since without this query H_p is random to $\mathcal{Z}$ and, crucially, $\mathsf{SIM}_{\mathsf{OWF}}$ works in a non-programmable RO model, and in particular has a read-only access to H_p queries and hence has no information about $\overline{\mathsf{hpw}}$ if $\mathcal{Z}$ doesn't perform this query, this modification is indistinguishable unless $\mathcal{Z}$ guesses H_p on some query to $\mathsf{S}^{\mathsf{ckid}}$, leading to the following:

$$\Delta_{\mathcal{Z}}(\mathsf{G}_{11}, \mathsf{G}_{12}) \leq q_{\mathsf{ses}}/|\mathcal{HP}| \tag{25}$$

Game G_{13} (ideal world): Game G_{13} just renders game G_{12} in a Figure 14, i.e.

$$\Delta_{\mathcal{Z}}(\mathsf{G}_{12}, \mathsf{G}_{13}) = 0 \tag{26}$$

We argue that game G_{13} also renders the ideal-world interaction created by $\mathcal{F}_{\mathsf{TFA\text{-}KE}}$ and our simulator SIM, and we explain this for each oracle call below:

- $\mathsf{Init}^{\mathsf{sid}}$ models a (StoreFile, sid, S) call to S. In the ideal-world execution, upon receiving (StoreFile, sid, S) from $\mathcal{F}_{\mathsf{TFA\text{-}KE}}$, SIM queries $\mathsf{SIM}_{\mathsf{OWF}}$ to obtain the same C_F as $\mathsf{Init}^{\mathsf{sid}}$, and sends it via the $\mathcal{F}_{\mathsf{NIZK}}$ interface, through which $\mathcal{Z}$ learns C_F. After receiving π, the tuple $((\mathsf{sid}, C_F), \perp, \pi)$ is recorded.
- $\mathsf{O}^{\mathsf{sid}}_{\mathsf{compr}}$ models a CompromiseFile call to S. In the ideal-world execution, upon receiving (LeakVal, sid) from $\mathsf{SIM}_{\mathsf{OWF}}$, SIM sends (CompromiseFile, sid) to $\mathcal{F}_{\mathsf{TFA\text{-}KE}}$, marks sid as compromised, selects the same v_1 and k as $\mathsf{O}^{\mathsf{sid}}_{\mathsf{compr}}$, and aborts with $[abort.key]$ if $\mathsf{H}^{\mathsf{sid}}_{\mathsf{code}}$ was previously queried for some T. Then, SIM reveals (k, F, C_F, d_F, π) to $\mathcal{Z}$. Note that C_F and π are omitted by $\mathsf{O}^{\mathsf{sid}}_{\mathsf{compr}}$ since $\mathcal{Z}$ already learns them through interaction with $\mathsf{Init}^{\mathsf{sid}}$.
- $\mathsf{H}^{[\mathsf{sid}]}_{\mathsf{name}}$ models calls to the hash functions $\mathsf{H}_{\mathsf{code}}$ and H_p. In the ideal-world execution, upon a query (k, T) to $\mathsf{H}^{\mathsf{sid}}_{\mathsf{code}}$, SIM services it as $\mathcal{F}_{\mathsf{RO}}$, except when sid is compromised and $k^{=}k$, in which case it programs $\mathsf{H}^{\mathsf{sid}}_{\mathsf{code}}(k, T) := v_1 - \Delta_T$, where Δ_T is $\mathcal{F}_{\mathsf{TFA\text{-}KE}}$'s response to (EvalCodeDiff, sid, $1, T$). By definition, $\Delta_T = \mathsf{R}^{\mathsf{sid}}(T) - \mathsf{R}^{\mathsf{sid}}(1)$, where $\mathsf{R}^{\mathsf{sid}}$ is a random function maintained by $\mathcal{F}_{\mathsf{TFA\text{-}KE}}$. Thus, $\mathsf{H}^{\mathsf{sid}}_{\mathsf{code}}(k, T)$ is programmed as $v1 - (\mathsf{R}^{\mathsf{sid}}(T) - \mathsf{R}^{\mathsf{sid}}(1)) = v1 + \mathsf{R}^{\mathsf{sid}}(1) - \mathsf{R}^{\mathsf{sid}}(T)$, which is the value returned by $\mathsf{H}^{[\mathsf{sid}]}_{\mathsf{name}}(x)$. For a query pw to H_p, SIM services it as $\mathcal{F}_{\mathsf{RO}}$ with bookkeeping, except in the case of a collision, in which case SIM aborts with $[abort.hash]$, as in $\mathsf{H}^{[\mathsf{sid}]}_{\mathsf{name}}(x)$.
- $\mathsf{D}^{\mathsf{sid}}_{\mathsf{code}}$ models code retrieval via a GetCode call to D. In the ideal-world execution, upon receiving (GetCode, sid, T) from D, $\mathcal{F}_{\mathsf{TFA\text{-}KE}}$ sends $\mathsf{R}^{\mathsf{sid}}(T)$ to D, which is exactly what $\mathsf{D}^{\mathsf{sid}}_{\mathsf{code}}(T)$ returns.
- $\mathsf{C}^{\mathsf{ckid}}_1$ models the first round of the C's code. In the ideal-world execution, upon receiving (CSession, sid, ssid, S, pw, code), $\mathcal{F}_{\mathsf{TFA\text{-}KE}}$ records $\langle$ssid, C, S, pw, code$\rangle$ if it is the first CSession query for ssid. Then, upon receiving (CSession, sid, ssid, C, S) from $\mathcal{F}_{\mathsf{TFA\text{-}KE}}$, SIM obtains (C'_F, π') and marks session $\mathsf{C}^{\mathsf{ckid}}$ as fresh. Since nothing new is revealed to $\mathcal{Z}$, $\mathsf{C}^{\mathsf{ckid}}_1$ simply stores $(\mathsf{pw}_\mathsf{C}, \mathsf{code}, C'_F, \pi')$ and marks $\mathsf{C}^{\mathsf{ckid}}$ as fresh.

– $\mathsf{S}^{\mathsf{ckid}}$ models the S's code. In the ideal-world execution, upon receiving (SSession, $\mathsf{sid}, \mathsf{ssid}, \mathsf{S}, \mathsf{C}$) from $\mathcal{F}_{\mathsf{TFA\text{-}KE}}$, SIM sends (C_F, π) to $\mathcal{A}^*$, marks session $\mathsf{S}^{\mathsf{ckid}}$ as fresh, and sends (Seval, $\mathsf{ckid}, \mathsf{C}, (\mathsf{sid}, C_F)$) as if from $\mathcal{F}_{\mathsf{CKEM}}$ to $\mathcal{A}^*$. These values are omitted since $\mathcal{Z}$ already knows C_F and π. If $\mathcal{A}^*$ then sends (Sout, ckid) to $\mathcal{F}_{\mathsf{CKEM}}$, SIM marks $\mathsf{S}^{\mathsf{ckid}}$ as $\mathsf{completed}$ and sends (NewKey, $\mathsf{sid}, \mathsf{ssid}, \mathsf{S}, \perp$) to $\mathcal{F}_{\mathsf{TFA\text{-}KE}}$, which in turn sends a uniformly random SK_S to S. This corresponds to the $b = 0$ branch in $\mathsf{S}^{\mathsf{ckid}}$.
If instead $\mathcal{A}^*$ sends (AdvCeval, ckid, x) to $\mathcal{F}_{\mathsf{CKEM}}$, SIM first checks whether $\mathsf{sid}' \neq \mathsf{sid}$ or $(\cdot, \mathsf{hpw}) \notin \mathsf{T}_\mathsf{p}$. If so, the test fails immediately. Otherwise, it queries (TestPwd, $\mathsf{sid}, \mathsf{ssid}, \mathsf{S}, \mathsf{pw}, s_\mathsf{C}$) at $\mathcal{F}_{\mathsf{TFA\text{-}KE}}$ and fails the test if it is a "wrong guess". In that case, SIM sends (NewKey, $\mathsf{sid}, \mathsf{ssid}, \mathsf{S}, \perp$) to $\mathcal{F}_{\mathsf{TFA\text{-}KE}}$ and (Cout, $\mathsf{ssid}, \perp$) to $\mathcal{A}^*$ via $\mathcal{F}_{\mathsf{CKEM}}$. Thus, $\mathcal{Z}$ learns $(\perp, \mathsf{SK}_\mathsf{S})$, where $\perp$ is returned by $\mathcal{F}_{\mathsf{CKEM}}$ and SK_S is the server's output.
The only non-failing case is when the TestPwd query to $\mathcal{F}_{\mathsf{TFA\text{-}KE}}$ succeeds, which occurs when $(\mathsf{pw}, s_\mathsf{C})$ matches the record stored by $\mathcal{F}_{\mathsf{TFA\text{-}KE}}$. This means $\mathsf{hpw} = \mathsf{H}_\mathsf{p}(\overline{\mathsf{pw}})$ and $s_\mathsf{C} = \mathsf{R}^{\mathsf{sid}}(T)$, which is the same condition checked in $\mathsf{S}^{\mathsf{ckid}}$. In this case, SIM sends (NewKey, $\mathsf{sid}, \mathsf{ssid}, \mathsf{S}, \mathsf{SK}$) to $\mathcal{F}_{\mathsf{TFA\text{-}KE}}$ and (Cout, $\mathsf{ssid}, \mathsf{SK}$) to $\mathcal{A}^*$ via $\mathcal{F}_{\mathsf{CKEM}}$, so $\mathcal{Z}$ learns $(\mathsf{SK}_\mathsf{S}, \mathsf{SK}_\mathsf{S})$.
– $\mathsf{C}_2^{\mathsf{ckid}}$ models the second round of the C's code. It first checks the witness. When $b = 0$, it models the honest path where (Cout, ckid) is called, and $\mathsf{SK}_\mathsf{C} := \mathsf{SK}_\mathsf{S}$ is returned iff $C'_F = C_F$, $(\mathsf{pw}_\mathsf{C}, \mathsf{code}) = (\overline{\mathsf{pw}}, \mathsf{R}^{\mathsf{sid}}(T))$, and $\mathsf{S}^{\mathsf{ckid}}$ is marked $\mathsf{completed}(T, \mathsf{SK}_\mathsf{S})$. Otherwise, SK_C is uniformly random.
When $b = 1$, it models the adversarial path where (AdvSeval($\mathsf{ckid}, y, \mathsf{SK}^*$)) is called. In the ideal-world execution, SIM also first checks the witness $C'_F = \mathsf{COM}(f^*(\mathsf{sid}, \mathsf{hpw}, s); {d_F}')$, exactly as in $\mathsf{C}_2^{\mathsf{ckid}}$. If the check fails, SIM short-circuits to the end and sends (RKey, $\mathsf{sid}, \mathsf{ssid}, \mathsf{C}$) to $\mathcal{F}_{\mathsf{TFA\text{-}KE}}$, which yields a uniformly random SK_C.
After verifying that $\mathsf{S}^{\mathsf{ckid}}$ is marked $\mathsf{completed}$, it checks whether $C'_F = C_F$. If not, it sends (RKey, $\mathsf{sid}, \mathsf{ssid}, \mathsf{C}$) to $\mathcal{F}_{\mathsf{TFA\text{-}KE}}$. Otherwise, it sends (NewKey, $\mathsf{sid}, \mathsf{ssid}, \mathsf{C}, \perp$) to $\mathcal{F}_{\mathsf{TFA\text{-}KE}}$, mirroring $\mathsf{C}_2^{\mathsf{ckid}}$.
Upon receiving (AdvSeval, $\mathsf{ckid}, y, \mathsf{SK}^*$), after parsing y, it checks if $(\mathsf{sid}', C'_F) \neq (\mathsf{sid}, \mathsf{COM}(F'; {d_F}'))$. If so, it sends (RKey, $\mathsf{sid}, \mathsf{ssid}, \mathsf{C}$) to $\mathcal{F}_{\mathsf{TFA\text{-}KE}}$, yielding a uniformly random SK_C, just like $\mathsf{C}_2^{\mathsf{ckid}}$.
Otherwise, it checks whether $C'_F = C_F$. If not, and sid is not $\mathsf{compromised}$, it aborts with $[abort.cmt]$, as in $\mathsf{C}_2^{\mathsf{ckid}}$. If sid is $\mathsf{compromised}$, SIM sends (Impersonate, $\mathsf{sid}, \mathsf{ssid}, 1, v_1 - s_\mathsf{S}$) and (NewKey, $\mathsf{sid}, \mathsf{ssid}, \mathsf{C}, \mathsf{SK}$) to $\mathcal{F}_{\mathsf{TFA\text{-}KE}}$. The Impersonate call checks whether $\mathsf{pw}_\mathsf{C} = \overline{\mathsf{pw}}$ and $\mathsf{code} = \mathsf{R}^{\mathsf{sid}}(1) + v_1 - s_\mathsf{S}$. If true, the subsequent NewKey call yields $\mathsf{SK}_\mathsf{C} := \mathsf{SK}^*$; otherwise, it yields a uniformly random SK_C.
If $C'_F \neq C_F$, SIM searches T_p for a pw such that $(\mathsf{pw}, \mathsf{hpw}) \in \mathsf{T}_\mathsf{p}$, which corresponds to having $\mathsf{hpw}' = \mathsf{H}_\mathsf{p}(\mathsf{pw}_\mathsf{C})$ in $\mathsf{C}_2^{\mathsf{ckid}}$. If it finds such a pw, it sends (TestPwd, $\mathsf{sid}, \mathsf{ssid}, \mathsf{C}, \mathsf{pw}, s - s_\mathsf{S}$) and (NewKey, $\mathsf{sid}, \mathsf{ssid}, \mathsf{C}, \mathsf{SK}^*$) to $\mathcal{F}_{\mathsf{TFA\text{-}KE}}$. The TestPwd call returns "correct guess" and marks the record as $\mathsf{compromised}$ when $\mathsf{code} = s' - s_\mathsf{S}$. In that case, the subsequent NewKey yields $\mathsf{SK}_\mathsf{C} := \mathsf{SK}^*$. Otherwise, it yields a uniformly random SK_C, just like in $\mathsf{C}_2^{\mathsf{ckid}}$.

– $\mathsf{O}_{\mathsf{group}}$ models $\mathcal{Z}$'s group operations. In the ideal-world execution, upon receiving $(\text{ADVTEST}, \mathsf{sid}, (\mathsf{pw}, s))$ from $\mathsf{SIM}_{\mathsf{OWF}}$, SIM simply sends $(\text{OFFLINETEST}, \mathsf{sid}, \mathsf{pw}, 1, s^{!}-!v_1)$ to $\mathcal{F}_{\mathsf{TFA\text{-}KE}}$ and forwards the result back to $\mathsf{SIM}_{\mathsf{OWF}}$. Here, the OFFLINETEST query corresponds to checking whether $(\mathsf{pw}, s^{-}v_1) = (\overline{\mathsf{pw}}, \mathsf{R}^{\mathsf{sid}}(1))$, which is exactly what $\mathsf{O}_{\mathsf{group}}$ does.

Summarizing the above explanations we conclude:

$$\mathsf{Exp}_{\mathcal{Z}}[\mathsf{G}_{13}] = \mathbf{Ideal}_{\mathcal{Z}}[\mathcal{F}_{\mathsf{TFA\text{-}KE}}, \mathsf{SIM}] \tag{27}$$

Summing up all the above bounds we conclude that for an efficient $\mathcal{Z}$ its real-vs-ideal distinguishing advantage, $|\mathbf{Real}_{\mathcal{Z}}[\mathsf{TFA}_{\mathsf{gen}}] - \mathbf{Ideal}_{\mathcal{Z}}[\mathcal{F}_{\mathsf{TFA\text{-}KE}}, \mathsf{SIM}]|$, is negligible under the stated assumptions, which proves the theorem.

F Non-Interactive Zero-Knowledge Proofs

In this section we discuss the NIZKs used by our protocols $\mathsf{TFA}_{\mathsf{gen}}$, $\mathsf{CKEM}_{\mathsf{Vand}}$, and $\mathsf{CKEM}_{\mathsf{PoNR}}$. We define universally composable NIZKs by ideal functionality $\mathcal{F}_{\mathsf{NIZK}}$ (Figure 1), which is parameterized by a relation between statements and witnesses.

F.1 $\mathsf{CKEM}_{\mathsf{Vand}}$ and $\mathsf{TFA}_{\mathsf{gen}}$ with f_{Vand}

Protocol $\mathsf{CKEM}_{\mathsf{Vand}}$ (Section 5.3) uses realizations of $\mathcal{F}_{\mathsf{NIZK}}$ for two relations: $\mathcal{R}_{\mathsf{C}}$ and $\mathcal{R}_{\mathsf{S}}$. As discussed in Section 5.3, the former enforces the honest behavior of clients and the latter enforces the honest behavior of servers. We define both relations formally here.

$\mathcal{R}_{\mathsf{C}}$ is the set $\{(\mathbb{x}, \mathbb{w})\}$ of statement-witness pairs, where statement $\mathbb{x} = (\mathsf{sid}, \mathsf{pk}_{\mathsf{C}}, \boldsymbol{G}) \in \{0,1\}^* \times \mathbb{G} \times (\mathbb{G}^2)^{N+1}$, witness $\mathbb{w} = (\mathsf{hpw}, s_{\mathsf{C}}, \boldsymbol{d}_G) \in \{0,1\}^{\lfloor \log p \rfloor - t - 1} \times \{0,1\}^t \times \mathbb{Z}_p^{N+1}$, and $\mathbb{x}$ and $\mathbb{w}$ satisfy:

$$\forall_{j=0}^{N}: \quad G_j = (g^{d_{G_j}}, {\mathsf{pk}_{\mathsf{C}}}^{d_{G_j}} \cdot \prod_{i=\max\{j,1\}}^{N} \mathsf{H}_{\mathbb{G}}^{\mathsf{sid}}(i)^{\binom{i}{j}(\mathsf{hpw}|0|s_{\mathsf{C}})^{i-j}})$$

$\mathcal{R}_{\mathsf{S}}$ is the set $\{(\mathbb{x}, \mathbb{w})\}$ of statement-witness pairs, where statement $\mathbb{x} = (C_F, \mathsf{pk}_{\mathsf{C}}, \boldsymbol{G}, C_K, (V_0', V_1')) \in \mathbb{G}^3 \times \mathbb{G} \times (\mathbb{G}^2)^{N+1} \times \mathbb{G}^3 \times (\mathbb{G}^2)^2$, witness $\mathbb{w} = (F, d_F, s_{\mathsf{S}}, (r_0, r_1), K, d_K, (d_{V_0}, d_{V_1}), (V_0, V_1)) \in \mathbb{G} \times \mathbb{Z}_p^2 \times \{0,1\}^t \times \mathbb{Z}_p^2 \times \mathbb{G} \times \mathbb{Z}_p^2 \times \mathbb{Z}_p^2 \times (\mathbb{G}^2)^2$, and $\mathbb{x}$ and $\mathbb{w}$ satisfy:

$$C_F = (g^{d_{F,0}}, g^{d_{F,1}}, F \cdot g^{d_{F,0} d_{F,1}}) \ \wedge \ C_K = (g^{d_{K,0}}, g^{d_{K,1}}, K \cdot g^{d_{K,0} d_{K,1}}) \ \wedge \ r_0, r_1 \neq 0$$
$$V_0 = (g^{d_{V_0}} \cdot (\textstyle\prod_{j=0}^{N} (G_{j,0})^{s_{\mathsf{S}}^j})^{r_0}, {\mathsf{pk}_{\mathsf{C}}}^{d_{V_0}} \cdot K \cdot (\textstyle\prod_{j=0}^{N} (G_{j,1})^{s_{\mathsf{S}}^j} / F)^{r_0})$$
$$V_1 = (g^{d_{V_1}} \cdot (\textstyle\prod_{j=0}^{N} (G_{j,0})^{(s_{\mathsf{S}} - 2^t)^j})^{r_1}, {\mathsf{pk}_{\mathsf{C}}}^{d_{V_1}} \cdot K \cdot (\textstyle\prod_{j=0}^{N} (G_{j,1})^{(s_{\mathsf{S}} - 2^t)^j} / F)^{r_1})$$
$$(V_0', V_1') = (V_0, V_1) \ \vee \ (V_0', V_1') = (V_1, V_0)$$

In addition to our realization of $\mathcal{F}_{\mathsf{CKEM}}$ for f_{Vand}, instantiating the generic $\mathsf{TFA}_{\mathsf{gen}}$ construction for OWF f_{Vand} also requires a realization of $\mathcal{F}_{\mathsf{NIZK}}[\mathcal{R}_{\mathsf{Init}}]$ for OWF f_{Vand} and commitment scheme EIG. $\mathcal{R}_{\mathsf{Init}}$ is defined in Section 5.2, but for clarity we repeat it here for this particular OWF and commitment scheme.

$\mathcal{R}_{\mathsf{Init}}$ is the set $\{(\mathbb{x}, \mathbb{w})\}$ of statement-witness pairs, where statement $\mathbb{x} = (\mathsf{sid}, C_F) \in \{0,1\}^* \times \mathbb{G}^3$, witness $\mathbb{w} = (\overline{\mathsf{hpw}}, s, d_F) \in \{0,1\}^{\lfloor \log p \rfloor - t - 1} \times \{0,1\}^t \times \mathbb{Z}_p^2$, and $\mathbb{x}$ and $\mathbb{w}$ satisfy:

$$C_F = (g^{d_{F,0}}, g^{d_{F,1}}, g^{d_{F,0} d_{F,1}} \cdot \prod_{i \in [N]} \mathsf{H}^{\mathsf{sid}}_{\mathbb{G}}(i)^{(\overline{\mathsf{hpw}}|0|s)^i})$$

F.2 $\mathsf{CKEM}_{\mathsf{PoNR}}$ and $\mathsf{TFA}_{\mathsf{gen}}$ with f_{PoNR}

Protocol $\mathsf{CKEM}_{\mathsf{PoNR}}$ (Section 5.4) uses realizations of $\mathcal{F}_{\mathsf{NIZK}}$ for two relations: $\mathcal{R}_{\mathsf{C}}$ and $\mathcal{R}_{\mathsf{S}}$. As discussed in Section 5.4, the former enforces the honest behavior of clients and the latter enforces the honest behavior of servers. We define both relations formally here.

$\mathcal{R}_{\mathsf{C}}$ is the set $\{(\mathbb{x}, \mathbb{w})\}$ of statement-witness pairs, where statement $\mathbb{x} = (\mathsf{sid}, \mathsf{pk}_{\mathsf{C}}, \boldsymbol{G}, \hat{G}, \boldsymbol{C}_m) \in \{0,1\}^* \times \mathbb{G} \times (\mathbb{G}^2)^N \times \mathbb{G}^2 \times \mathbb{G}^{2t}$, witness $\mathbb{w} = (\mathsf{hpw}, s_{\mathsf{C}}, \boldsymbol{d}_G, d_{\hat{G}}, \boldsymbol{\delta}, \boldsymbol{d}_m) \in \mathbb{Z}_p^N \times \{0,1\}^t \times \mathbb{Z}_p^N \times \mathbb{Z}_p \times (\mathbb{Z}_p^t)^N \times \mathbb{Z}_p^{2t}$, and $\mathbb{x}$ and $\mathbb{w}$ satisfy:

$$\begin{gathered}
\forall_{i \in [N]}: \quad G_i = (g^{d_{G_i}}, {\mathsf{pk}_{\mathsf{C}}}^{d_{G_i}} \cdot \mathsf{H}^{\mathsf{sid}}_{\mathbb{G}}(i)^{\mathsf{hpw}_i / \prod_{j \in [t]} \delta_{i,j}}) \\
\hat{G} = (g^{d_{\hat{G}}}, {\mathsf{pk}_{\mathsf{C}}}^{d_{\hat{G}}} \cdot \textstyle\prod_{i \in [N]} \hat{\mathsf{H}}^{\mathsf{sid}}_{\mathbb{G}}(i)^{\mathsf{hpw}_i}) \\
\forall_{j \in [t]}: \quad C_{m_{j,s_{\mathsf{C}}[j]}} = g^{d_{m_{j,s_{\mathsf{C}}[j]}}} \cdot \textstyle\prod_{i \in [N]} {h_i}^{\delta_{i,j}} \\
\forall_{j \in [t]}: \quad C_{m_{j,1-s_{\mathsf{C}}[j]}} = g^{d_{m_{j,1-s_{\mathsf{C}}[j]}}} \cdot \textstyle\prod_{i \in [N]} {h_i}^{\delta_{i,j} \cdot \mathsf{H}_{\times}(i,j)}
\end{gathered}$$

$\mathcal{R}_{\mathsf{S}}$ is the set $\{(\mathbb{x}, \mathbb{w})\}$ of statement-witness pairs, where statement $\mathbb{x} = (C_F, \mathsf{pk}_{\mathsf{C}}, \boldsymbol{G}, \hat{G}, \boldsymbol{C}_m, C_K, V) \in \mathbb{G}^3 \times \mathbb{G} \times (\mathbb{G}^2)^N \times \mathbb{G}^2 \times \mathbb{G}^{2t} \times \mathbb{G}^3 \times \mathbb{G}^2$, witness $\mathbb{w} = (F, d_F, s_{\mathsf{S}}, \boldsymbol{m}^{\mathsf{S}}, \boldsymbol{d}^{\mathsf{S}}_m, r, K, d_K, d_V) \in \mathbb{G} \times \mathbb{Z}_p^2 \times \{0,1\}^t \times (\mathbb{Z}_p^N)^t \times \mathbb{Z}_p^t \times \mathbb{Z}_p \times \mathbb{G} \times \mathbb{Z}_p^2 \times \mathbb{Z}_p$, and $\mathbb{x}$ and $\mathbb{w}$ satisfy:

$$\begin{gathered}
C_F = (g^{d_{F,0}}, g^{d_{F,1}}, F \cdot g^{d_{F,0} d_{F,1}}) \ \wedge \ C_K = (g^{d_{K,0}}, g^{d_{K,1}}, K \cdot g^{d_{K,0} d_{K,1}}) \ \wedge \ r \neq 0 \\
\forall_{j \in [t]}: \quad C_{m_{j,s_{\mathsf{S}}[j]}} = g^{d_{m_j}} \cdot \textstyle\prod_{i \in [N]} {h_i}^{m_{i,j}} \\
V = (g^{d_V} \cdot (\hat{G}_0 \cdot \textstyle\prod_{i \in [N]} (G_{i,0})^{\prod_{j \in [t]} m_{i,j}})^r, {\mathsf{pk}_{\mathsf{C}}}^{d_V} \cdot K \cdot (\hat{G}_1 \cdot \textstyle\prod_{i \in [N]} (G_{i,1})^{\prod_{j \in [t]} m_{i,j}} / F)^r)
\end{gathered}$$

In addition to our realization of $\mathcal{F}_{\mathsf{CKEM}}$ for f_{PoNR}, instantiating the generic $\mathsf{TFA}_{\mathsf{gen}}$ construction for OWF f_{PoNR} also requires a realization of $\mathcal{F}_{\mathsf{NIZK}}[\mathcal{R}_{\mathsf{Init}}]$ for OWF f_{PoNR} and commitment scheme EIG. $\mathcal{R}_{\mathsf{Init}}$ is defined in Section 5.2, but for clarity we repeat it here for this particular OWF and commitment scheme.

$\mathcal{R}_{\mathsf{Init}}$ is the set $\{(\mathbb{x}, \mathbb{w})\}$ of statement-witness pairs, where statement $\mathbb{x} = (\mathsf{sid}, C_F) \in \{0,1\}^* \times \mathbb{G}^3$, witness $\mathbb{w} = (\overline{\mathsf{hpw}}, s, d_F) \in \mathbb{Z}_p^N \times \{0,1\}^t \times \mathbb{Z}_p^2$, and $\mathbb{x}$ and $\mathbb{w}$ satisfy:

$$C_F = (g^{d_{F,0}}, g^{d_{F,1}}, g^{d_{F,0} d_{F,1}} \cdot \prod_{i \in [N]} \mathsf{H}^{\mathsf{sid}}_{\mathbb{G}}(i)^{\mathsf{hpw}_i \cdot \prod_{j \in [t], s[j]=1} \mathsf{H}_{\times}(i,j)} \cdot \hat{\mathsf{H}}^{\mathsf{sid}}_{\mathbb{G}}(i)^{\mathsf{hpw}_i})$$

F.3 Efficient Realizations

Our OWFs and CKEM protocols are specifically designed so that their requisite NIZKs can be efficiently instantiated using standard techniques. In particular, every relation is a discrete-logarithm representation over public bases. A Sigma protocol for each of these relations is a straightforward extension of Schnorr's proof of discrete-logarithm knowledge, see e.g. the report by Camenisch and Stadler [12]. Sigma protocols can be compiled into universally composable NIZK proof systems using the Fischlin transform, which is practically efficient [20,14].

Some relations for f_{Vand} require proving that witnesses are representable as bitstrings of a certain length. Such statements can be expressed as discrete-logarithm representations by committing to each bit individually, proving that each bit is either 0 or 1, and then proving that the committed witness is represented by those bits. OR-compilation of Sigma protocols, used by our other relations as well, is easily handled by a standard technique. Finally, the relations $\mathcal{R}_{\mathsf{S}}$ of both $\mathsf{CKEM}_{\mathsf{Vand}}$ and $\mathsf{CKEM}_{\mathsf{PoNR}}$ include conditions that certain values are nonzero. This is easily handled by proving the existence of a multiplicative inverse.

Proof size and verifier efficiency can be significantly improved by employing Sigma-protocol compression [3].

G Proof of $\mathsf{CKEM}_{\mathsf{Vand}}$ Security

In this section we prove Theorem 8, i.e. that protocol $\mathsf{CKEM}_{\mathsf{Vand}}$ realizes $\mathcal{F}_{\mathsf{CKEM}}[\mathsf{SSEM}[f^*_{\mathsf{Vand}}, \mathsf{EIG}], \mathsf{SSEM}_{\mathsf{Adv}}[f^*_{\mathsf{Vand}}]]$.

Specifically, for any efficient adversary $\mathcal{A}^*$ against protocol $\mathsf{CKEM}_{\mathsf{Vand}}$, there exists a simulator SIM such that no efficient environment $\mathcal{Z}$ can distinguish the view of $\mathcal{A}^*$ interacting with the real $\mathsf{CKEM}_{\mathsf{Vand}}$ protocol and the view of SIM interacting with the ideal functionality $\mathcal{F}_{\mathsf{CKEM}}[\mathsf{SSEM}[f^*_{\mathsf{Vand}}, \mathsf{EIG}], \mathsf{SSEM}_{\mathsf{Adv}}[f^*_{\mathsf{Vand}}]]$ with advantage better than $3 \cdot \mathsf{Adv}_{\mathsf{DDH}} + \frac{{q_{\mathsf{C}}}^2 + {q_{\mathsf{S}}}^2 + q_{\mathsf{S}} q_{\mathsf{H}} + q_{\mathsf{H}}}{p}$ where there are at most q_{C} client sessions, there are at most q_{S} server sessions, there are at most q_{H} queries to H_{d} and H_{SK}, and $\mathsf{Adv}_{\mathsf{DDH}}$ is a bound on the DDH advantage of any efficient algorithm.

Proof. For any adversary $\mathcal{A}^*$, we construct simulator $\mathsf{SIM}_{\mathsf{CKEM\text{-}Vand}}$ as shown in Figure 15. Without loss of generality, we assume that $\mathcal{A}^*$ is a "dummy" adversary that merely passes messages to and from the environment $\mathcal{Z}$.

We now show that, for any efficient (i.e. PPT) $\mathcal{Z}$, the distinguishing advantage of $\mathcal{Z}$ between the real and simulated worlds is negligible. We denote the probability that $\mathcal{Z}$ outputs 1 in the interaction with the real world game or the simulated game as resp. $\mathbf{Real}_{\mathcal{Z}}[\mathsf{CKEM}_{\mathsf{Vand}}]$ and $\mathbf{Ideal}_{\mathcal{Z}}[\mathcal{F}_{\mathsf{CKEM}}, \mathsf{SIM}_{\mathsf{CKEM\text{-}Vand}}]$. We denote the probability that $\mathcal{Z}$ outputs 1 in an interaction with security game G_i as $\mathsf{Exp}_{\mathcal{Z}}[\mathsf{G}_i] = \Pr[1 \leftarrow \mathcal{Z}^{\mathsf{G}_i}(1^\kappa)]$, and we define $\Delta(\mathsf{G}_i, \mathsf{G}_j) = |\mathsf{Exp}_{\mathcal{Z}}[\mathsf{G}_i] - \mathsf{Exp}_{\mathcal{Z}}[\mathsf{G}_j]|$. The argument proceeds by a series of game changes, starting with game G_0 s.t. $\mathsf{Exp}_{\mathcal{Z}}[\mathsf{G}_0] = \mathbf{Real}_{\mathcal{Z}}[\mathsf{CKEM}_{\mathsf{Vand}}]$ and ends with game G_{10} s.t. $\mathsf{Exp}_{\mathcal{Z}}[\mathsf{G}_{10}] = \mathbf{Ideal}_{\mathcal{Z}}[\mathcal{F}_{\mathsf{CKEM}}, \mathsf{SIM}_{\mathsf{CKEM\text{-}Vand}}]$. We must show that $\Delta(\mathsf{G}_0, \mathsf{G}_{10})$ is negligible.

- Program $\mathsf{H_d}$ and $\mathsf{H_{SK}}$ outputs uniformly at random, except where special $\mathsf{H_{SK}}$ programming is noted below.
- On $(\textsc{Ceval}, \mathsf{ssid_C}, \mathsf{C}, (\mathsf{sid_C}, C_F))$ from $\mathcal{F}_{\mathsf{CKEM}}$:
  - Sample $\mathsf{pk_C} \leftarrow_\$ \mathbb{G}$ and $\boldsymbol{G} \leftarrow_\$ (\mathbb{G}^2)^{N+1}$.
  - Send $(\textsc{Prove}, \mathsf{C}, \mathsf{ssid_C}, (\mathsf{sid_C}, \mathsf{pk_C}, \boldsymbol{G}))$ to $\mathcal{A}^*$ as if from $\mathcal{F}_{\mathsf{NIZK}}[\mathcal{R}_\mathsf{C}]$, await response $(\textsc{Proof}, \mathsf{ssid_C}, \pi_\mathsf{C})$, and save record $(\mathcal{R}_\mathsf{C}, (\mathsf{sid_C}, \mathsf{pk_C}, \boldsymbol{G}), \bot, \pi_\mathsf{C}, \mathsf{ssid_C})$.
  - Send $(\mathsf{sid_C}, \mathsf{pk_C}, \boldsymbol{G}, \pi_\mathsf{C})$ to $\mathcal{A}^*$ as if from C to S.
  - Await $(C_K, (V_0', V_1'), \pi_\mathsf{S})$ from $\mathcal{A}^*$ to C.
  - Define $\mathbb{x}_\mathsf{S} := (C_F, \mathsf{pk_C}, \boldsymbol{G}, C_K, (V_0', V_1'))$. If there exists no record $(\mathcal{R}_\mathsf{S}, \mathbb{x}_\mathsf{S}, \mathbb{w}_\mathsf{S}, \pi_\mathsf{S}, \mathsf{ssid_S})$ then send $(\textsc{Verify}, \mathsf{C}, \mathsf{ssid_C}, \mathbb{x}_\mathsf{S}, \pi_\mathsf{S})$ to $\mathcal{A}^*$ as if from $\mathcal{F}_{\mathsf{NIZK}}[\mathcal{R}_\mathsf{S}]$, await response $(\textsc{Witness}, \mathsf{ssid_C}, \mathbb{w}_\mathsf{S})$, and save record $(\mathcal{R}_\mathsf{S}, \mathbb{x}_\mathsf{S}, \mathbb{w}_\mathsf{S}, \pi_\mathsf{S}, \bot)$.
  - If $\mathbb{w}_\mathsf{S} \neq \bot$ then parse $\mathbb{w}_\mathsf{S} \to (F, d_F, s_\mathsf{S}, \cdot, K, d_K, \cdot, \cdot)$, and...
    - If $(\mathbb{x}_\mathsf{S}, \mathbb{w}_\mathsf{S}) \notin \mathcal{R}_\mathsf{S} \vee \mathsf{H_d^{sid_C}}(K) \neq d_K$: Send $(\textsc{AdvSeval}, \mathsf{ssid_C}, \bot, 0^\kappa)$ to $\mathcal{F}_{\mathsf{CKEM}}$.
    - Else: Send $(\textsc{AdvSeval}, \mathsf{ssid_C}, (\mathsf{sid_C}, F, s_\mathsf{S}, d_F), \mathsf{H_{SK}^{sid_C}}(\mathsf{ssid_C}, K))$ to $\mathcal{F}_{\mathsf{CKEM}}$.
  - If $\mathbb{w}_\mathsf{S} = \bot$, then...
    - If $\mathsf{ssid_C} \neq \mathsf{ssid_S}$: Send $(\textsc{AdvSeval}, \mathsf{ssid_C}, \bot, 0^\kappa)$ to $\mathcal{F}_{\mathsf{CKEM}}$.
    - Else: Send $(\textsc{Cout}, \mathsf{ssid_C})$ to $\mathcal{F}_{\mathsf{CKEM}}$.
- On $(\textsc{Seval}, \mathsf{ssid_S}, \mathsf{S}, (\mathsf{sid_S}, C_F))$ from $\mathcal{F}_{\mathsf{CKEM}}$:
  - Await $(\mathsf{sid_C}, \mathsf{pk_C}, \boldsymbol{G}, \pi_\mathsf{C})$ from $\mathcal{A}^*$ to S.
  - Define $\mathbb{x}_\mathsf{C} := (\mathsf{sid_S}, \mathsf{pk_C}, \boldsymbol{G})$. If there exists no record $(\mathcal{R}_\mathsf{C}, \mathbb{x}_\mathsf{C}, \mathbb{w}_\mathsf{C}, \pi_\mathsf{C}, \mathsf{ssid_C})$ then send $(\textsc{Verify}, \mathsf{S}, \mathsf{ssid_S}, \mathbb{x}_\mathsf{C}, \pi_\mathsf{C})$ to $\mathcal{A}^*$ as if from $\mathcal{F}_{\mathsf{NIZK}}[\mathcal{R}_\mathsf{C}]$, await response $(\textsc{Witness}, \mathsf{ssid_S}, \mathbb{w}_\mathsf{C})$, and save record $(\mathcal{R}_\mathsf{C}, \mathbb{x}_\mathsf{C}, \mathbb{w}_\mathsf{C}, \pi_\mathsf{C}, \bot)$.
  - If $\mathsf{sid_C} \neq \mathsf{sid_S} \vee (\mathbb{w}_\mathsf{C} \neq \bot \wedge (\mathbb{x}_\mathsf{C}, \mathbb{w}_\mathsf{C}) \notin \mathcal{R}_\mathsf{C})$: Send $(\textsc{AdvCeval}, \mathsf{ssid_S}, \bot)$ to $\mathcal{F}_{\mathsf{CKEM}}$, receive response $(\textsc{Cout}, \mathsf{ssid_S}, \bot)$, and break (i.e. don't run the following code).
  - If $\mathbb{w}_\mathsf{C} \neq \bot$: Parse $\mathbb{w}_\mathsf{C} \to (\mathsf{hpw}, s_\mathsf{C}, \cdot)$, send $(\textsc{AdvCeval}, \mathsf{ssid_S}, (\mathsf{sid_S}, \mathsf{hpw}, s_\mathsf{C}))$ to $\mathcal{F}_{\mathsf{CKEM}}$, and receive response $(\textsc{Cout}, \mathsf{ssid_S}, \mathsf{SK})$.
  - If $\mathbb{w}_\mathsf{C} = \bot$ then define $\mathsf{SK} := \bot$, and...
    - If $\mathsf{ssid_S} \neq \mathsf{ssid_C}$: Send $(\textsc{AdvCeval}, \mathsf{ssid_S}, \bot)$ to $\mathcal{F}_{\mathsf{CKEM}}$ and receive response $(\textsc{Cout}, \mathsf{ssid_S}, \bot)$.
    - Else: Send $(\textsc{Sout}, \mathsf{ssid_S})$ to $\mathcal{F}_{\mathsf{CKEM}}$.
  - Sample $K_0', K_1' \leftarrow_\$ \mathbb{G}$, and compute $V_0' \leftarrow \mathsf{EIG.Enc_{pk_C}}(K_0')$ and $V_1' \leftarrow \mathsf{EIG.Enc_{pk_C}}(K_1')$.
  - If $\mathsf{SK} \neq \bot$ then sample $\beta \leftarrow_\$ \{0,1\}$, compute $C_K \leftarrow \mathsf{EIG.Commit}(K_\beta'; \mathsf{H_d^{sid_S}}(K_\beta'))$, and program $\mathsf{H_{SK}^{sid_S}}(\mathsf{ssid_S}, K_\beta') := \mathsf{SK}$. Else $(\mathsf{SK} = \bot)$ sample $C_K \leftarrow_\$ \mathbb{G}^3$.
  - Define $\mathbb{x}_\mathsf{S} := (C_F, \mathsf{pk_C}, \boldsymbol{G}, C_K, (V_0', V_1'))$. Send $(\textsc{Prove}, \mathsf{S}, \mathsf{ssid_S}, \mathbb{x}_\mathsf{S})$ to $\mathcal{A}^*$ as if from $\mathcal{F}_{\mathsf{NIZK}}[\mathcal{R}_\mathsf{S}]$, await response $(\textsc{Proof}, \mathsf{ssid_S}, \pi_\mathsf{S})$, and save record $(\mathcal{R}_\mathsf{S}, \mathbb{x}_\mathsf{S}, \bot, \pi_\mathsf{S}, \mathsf{ssid_S})$.
  - Send $(C_K, (V_0', V_1'), \pi_\mathsf{S})$ to $\mathcal{A}^*$ as if from S to C.

Fig. 15. $\mathsf{SIM_{CKEM\text{-}Vand}}$: Simulator for protocol $\mathsf{CKEM_{Vand}}$

Game G_0 (the real world): The distinguisher $\mathcal{Z}$ interacts with $\mathsf{CKEM}_{\mathsf{Vand}}$ (Figure 7) in the role of the honest parties and in the role of the adversary. Functionality $\mathcal{F}_{\mathsf{NIZK}}$ (Figure 1) interacts with $\mathsf{CKEM}_{\mathsf{Vand}}$ and with $\mathcal{Z}$ via the adversarial interface. H_{d} and H_{SK} are true random oracles.

$$\mathsf{Exp}_{\mathcal{Z}}[\mathsf{G}_0] = \mathbf{Real}_{\mathcal{Z}}[\mathsf{CKEM}_{\mathsf{Vand}}]$$

Game G_1 (statements never collide): G_1 is G_0 with the following changes:

- If two C sessions ever pick the same pk_{C}, the game immediately aborts.
- If two S sessions ever pick the same K, the game immediately aborts.

Per the code of $\mathcal{F}_{\mathsf{NIZK}}$, it is possible for a single statement-proof pair $(\mathbb{x}, \pi)$ to correspond to more than one witness $\mathbb{w}$ iff statement $\mathbb{x}$ is proven more than once by honest parties. This game change ensures that the $\mathbb{x}_{\mathsf{C}}$ and $\mathbb{x}_{\mathsf{S}}$ statements proven by honest parties never repeat. Therefore, in future games, we can retrieve the one $\mathbb{w}$ for any $(\mathbb{x}, \pi)$ without ambiguity.

Every pk_{C} (resp. K) is sampled uniformly at random from $\mathbb{G}$. Therefore, the probability of a collision is upper-bounded by ${q_{\mathsf{C}}}^2/p$ (resp. ${q_{\mathsf{S}}}^2/p$). If no collision occurs, G_0 and G_1 are identical.

$$\Delta(\mathsf{G}_0, \mathsf{G}_1) \leq \frac{{q_{\mathsf{C}}}^2 + {q_{\mathsf{S}}}^2}{p}$$

Game G_2 (client NIZKs are simulated): G_2 is G_1 with only one change: $\mathbb{w}_{\mathsf{C}}$ is omitted when C calls $\mathcal{F}_{\mathsf{NIZK}}[\mathcal{R}_{\mathsf{C}}]$.Prove. Since honest clients never attempt to prove false statements, the check for $(\mathbb{x}, \mathbb{w}) \in \mathcal{R}_{\mathsf{C}}$ is simply skipped, and instead of a witness a special symbol $\mathbb{w} := \perp$ is recorded to represent that the produced proof is correct. The code of $\mathcal{F}_{\mathsf{NIZK}}[\mathcal{R}_{\mathsf{C}}]$.Verify is modified to always accept proofs for records with $\mathbb{w} = \perp$.

The views of $\mathcal{Z}$ in G_1 and G_2 are identical.

$$\Delta(\mathsf{G}_1, \mathsf{G}_2) = 0$$

Game G_3 (clients do not decrypt V_0' and V_1'): G_3 is G_2 with the following changes to C:

- Instead of sampling $(\mathsf{sk}_{\mathsf{C}}, \mathsf{pk}_{\mathsf{C}}) \leftarrow \mathsf{EIG.KG}$, only sample $\mathsf{pk}_{\mathsf{C}} \leftarrow_{\$} \mathbb{G}$.
- After verifying π_{S}, do not decrypt V_0' and V_1'. Instead, parse $\mathbb{w}_{\mathsf{S}} \rightarrow (F, \cdot, s_{\mathsf{S}}, \cdot, K, d_K, \cdot, \cdot)$. If $\mathsf{H}_{\mathsf{d}}^{\mathsf{sid}_{\mathsf{C}}}(K) = d_K \ \wedge \ F = f^*_{\mathsf{Vand}}(\mathsf{sid}_{\mathsf{C}}, \mathsf{hpw}, s_{\mathsf{C}} + s_{\mathsf{S}})$, then output $\mathsf{SK}' := \mathsf{H}_{\mathsf{SK}}^{\mathsf{sid}_{\mathsf{C}}}(\mathsf{ssid}_{\mathsf{C}}, K)$. Otherwise, output $\mathsf{SK}' \leftarrow_{\$} \{0,1\}^{\kappa}$.

Notice that C's changed code only runs if π_{S} verifies. $\mathcal{R}_{\mathsf{S}}$ guarantees that the received V_0', V_1', and C_K are consistent with correct server behavior on $\mathbb{w}_{\mathsf{S}}$, e.g. $C_K = \mathsf{EIG.Commit}(K; d_K)$. Therefore it is not necessary for C to refer to V_0', V_1', and C_K directly. In the previous game, C will decrypt K from either V_0' or V_1' iff $F = f^*_{\mathsf{Vand}}(\mathsf{sid}_{\mathsf{C}}, \mathsf{hpw}, s_{\mathsf{C}} + s_{\mathsf{S}})$, and if additionally $\mathsf{H}_{\mathsf{d}}^{\mathsf{sid}_{\mathsf{C}}}(K) = d_K$ then C will use

K to derive SK'. If either of those conditions does not occur, then C will output a random SK'. The behavior of C in this game is exactly the same.

The views of $\mathcal{Z}$ in G_2 and G_3 are identical.

$$\Delta(\mathsf{G}_2, \mathsf{G}_3) = 0$$

Game G_4 ($\boldsymbol{G}$ is random): G_4 is G_3 with only one change: instead of computing $(g_j)_{j\in\{0,1,\ldots,N\}}$ and $\boldsymbol{G} = (G_j)_{j\in\{0,1,\ldots,N\}}$ as in $\mathsf{CKEM}_{\mathsf{Vand}}$, C simply samples $\boldsymbol{G} \leftarrow_{\$} (\mathbb{G}^2)^{N+1}$.

Recall that client NIZKs are simulated, so it is not a problem that $\mathbb{x}_{\mathsf{C}}$ no longer has a correct witness. Additionally, the responses (V_0', V_1') that S computes from $\boldsymbol{G}$ are not decrypted and used by C, so it is not a problem that these responses are formed from random G ciphertexts.

Given an environment $\mathcal{Z}$ that can distinguish G_3 from G_4, we construct a reduction $\mathcal{R}$ that can solve DDH. The input to $\mathcal{R}$ is tuple $(A, B, C) \in \mathbb{G}^3$. $\mathcal{R}$ emulates game G_3 to $\mathcal{Z}$ with the following modifications to the code of C:

- Instead of sampling $\mathsf{pk}_{\mathsf{C}} \leftarrow_{\$} \mathbb{G}$, $\mathcal{R}$ samples $\delta \leftarrow_{\$} \mathbb{Z}_p$ and sets $\mathsf{pk}_{\mathsf{C}} := A \cdot g^{\delta}$.
- Instead of encrypting by $d_{G_j} \leftarrow_{\$} \mathbb{Z}_p$, $G_j \leftarrow \mathsf{ElG.Enc}_{\mathsf{pk}_{\mathsf{C}}}(g_j; d_{G_j})$, $\mathcal{R}$ samples $\mu, \nu \leftarrow_{\$} \mathbb{Z}_p$ and sets $G_j := (g^{\mu} \cdot B^{\nu}, g_j \cdot (A^{\mu} \cdot C^{\nu}) \cdot (g^{\mu} \cdot B^{\nu})^{\delta})$.

Here we apply the well-known DDH randomized self-reduction [44,39]. If (A, B, C) is a Diffie-Hellman tuple, then $(A, g^{\mu} \cdot B^{\nu}, A^{\mu} \cdot C^{\nu})$ is a random Diffie-Hellman tuple. If not, then it is a uniformly random tuple. For every C session, we apply a "shift" of δ to get a freshly random pk_{C}. If (A, B, C) is a Diffie-Hellman tuple, then each G_j ciphertext is a correct ElGamal encryption of g_j under pk_{C}, so the view of $\mathcal{Z}$ is identical to that in G_3. If (A, B, C) is not a Diffie-Hellman tuple, then each G_j ciphertext is independently random, so the view of $\mathcal{Z}$ is identical to that in G_4.

Reduction $\mathcal{R}$ outputs 1 iff $\mathcal{Z}$ outputs 1. Thus, the advantage of $\mathcal{Z}$ at distinguishing between G_3 and G_4 is equal to the advantage of $\mathcal{R}$ in the DDH game, which is $\leq \mathsf{Adv}_{\mathsf{DDH}}$.

$$\Delta(\mathsf{G}_3, \mathsf{G}_4) \leq \mathsf{Adv}_{\mathsf{DDH}}$$

Game G_5 (server NIZKs are simulated): G_5 is G_4 with the following changes:

- $\mathbb{w}_{\mathsf{S}}$ is truncated to $(F, \cdot, s_{\mathsf{S}}, \cdot, K, \cdot, \cdot, \cdot)$ when S calls $\mathcal{F}_{\mathsf{NIZK}}[\mathcal{R}_{\mathsf{S}}]$.Prove. Since honest servers never attempt to prove false statements, the check for $(\mathbb{x}, \mathbb{w}) \in \mathcal{R}_{\mathsf{S}}$ is simply skipped, and the code of $\mathcal{F}_{\mathsf{NIZK}}[\mathcal{R}_{\mathsf{S}}]$.Verify is modified to always accept proofs with witnesses of this truncated form.
- In the previous game, C parses F, s_{S}, K, and d_K from $\mathbb{w}_{\mathsf{S}}$ (see G_3), but d_K is now missing from the recorded witnesses of honestly generated server proofs. However, d_K is only used to check whether $\mathsf{H}_{\mathsf{d}}^{\mathsf{sid}_{\mathsf{C}}}(K) = d_K$, and this condition always holds for honest servers, so the code of C is modified to simply skip that check when handling a witness of this truncated form.

The views of $\mathcal{Z}$ in G_4 and G_5 are identical.

$$\Delta(\mathsf{G}_4, \mathsf{G}_5) = 0$$

Game G_6 (V_0' and V_1' are random when responding to unsuccessful dishonest clients): G_6 is G_5 with the following changes to S after verifying π_{C} and checking that $\mathsf{sid}_{\mathsf{C}} = \mathsf{sid}_{\mathsf{S}}$:

- If $\mathbb{w}_{\mathsf{C}} = \perp$ (i.e. π_{C} was generated by an honest C, see G_2), then proceed as before.
- Otherwise, parse $\mathbb{w}_{\mathsf{C}} \to (\mathsf{hpw}, s_{\mathsf{C}}, \cdot)$. If $F = f^*_{\mathsf{Vand}}(\mathsf{sid}_{\mathsf{S}}, \mathsf{hpw}, s_{\mathsf{C}} + s_{\mathsf{S}})$, then still proceed as before.
- Otherwise, do not compute V_0' and V_1' as normal. Instead, simply sample $V_0', V_1' \leftarrow_{\$} \mathbb{G}^2$.

Recall that server NIZKs are simulated, so it is not a problem that S no longer knows a witness for $\mathbb{x}_{\mathsf{S}}$. Furthermore, we are only modifying the behavior of S in the case that $\mathbb{w}_{\mathsf{C}} \neq \perp$, in which case $\boldsymbol{G}$ must be correctly formed for $\mathsf{sid}_{\mathsf{S}}, \mathsf{hpw}, s_{\mathsf{C}}$ (by $\mathcal{F}_{\mathsf{NIZK}}[\mathcal{R}_{\mathsf{C}}]$ soundness). Therefore, the algebra underlying $\mathsf{CKEM}_{\mathsf{Vand}}$ implies that, if $F \neq f^*_{\mathsf{Vand}}(\mathsf{sid}_{\mathsf{S}}, \mathsf{hpw}, s_{\mathsf{C}} + s_{\mathsf{S}})$, then V_0' and V_1' are two uniformly random ElGamal ciphertexts.

The views of $\mathcal{Z}$ in G_5 and G_6 are identical.

$$\Delta(\mathsf{G}_5, \mathsf{G}_6) = 0$$

Game G_7 (V_0' and V_1' are random when responding to honest clients): G_7 is G_6 with only one change: if $\mathbb{w}_{\mathsf{C}} = \perp$, then S samples $V_0', V_1' \leftarrow_{\$} \mathbb{G}^2$ instead of computing V_0' and V_1' as before.

Unlike the previous game change, this one does not result in an identical view. Nevertheless, since S should be computing V_0' and V_1' under a public key pk_{C} whose secret key is unknown to $\mathcal{Z}$, it should not be possible for $\mathcal{Z}$ to distinguish the replacement of V_0' and V_1' with random ciphertexts.

Given an environment $\mathcal{Z}$ that can distinguish G_6 from G_7, we construct a reduction $\mathcal{R}$ that can solve DDH. The input to $\mathcal{R}$ is tuple $(A, B, C) \in \mathbb{G}^3$. $\mathcal{R}$ emulates game G_6 to $\mathcal{Z}$ with the following modifications:

- In C, instead of sampling $\mathsf{pk}_{\mathsf{C}} \leftarrow_{\$} \mathbb{G}$, $\mathcal{R}$ samples $\delta \leftarrow_{\$} \mathbb{Z}_p$ and sets $\mathsf{pk}_{\mathsf{C}} := A \cdot g^{\delta}$. δ is recorded in association with proof π_{C}.
- In S, instead of encrypting by $d_{V_i} \leftarrow_{\$} \mathbb{Z}_p$, $V_i \leftarrow \mathsf{ElG.Enc}_{\mathsf{pk}_{\mathsf{C}}}(K; d_{V_i}) \cdot (\cdots)$ (for $i \in \{0, 1\}$), $\mathcal{R}$ samples $\mu_i, \nu_i \leftarrow_{\$} \mathbb{Z}_p$ and sets $V_i := (g^{\mu_i} \cdot B^{\nu_i}, K \cdot (A^{\mu_i} \cdot C^{\nu_i}) \cdot (g^{\mu_i} \cdot B^{\nu_i})^{\delta}) \cdot (\cdots)$, where δ is as recorded for π_{C}.

Again we utilize DDH randomized self-reduction with a random δ "shift" applied to each pk_{C}. If (A, B, C) is a Diffie-Hellman tuple, then the V_0, V_1 ciphertexts (and consequently the V_0', V_1' ciphertexts) are formed exactly as in $\mathsf{CKEM}_{\mathsf{Vand}}$ and the view of $\mathcal{Z}$ is identical to that in G_6. If (A, B, C) is not a Diffie-Hellman tuple, then the V_0, V_1 ciphertexts (and consequently the V_0', V_1' ciphertexts) are independently random and the view of $\mathcal{Z}$ is identical to that in G_7.

Reduction $\mathcal{R}$ outputs 1 iff $\mathcal{Z}$ outputs 1. Thus the advantage of $\mathcal{Z}$ at distinguishing between G_6 and G_7 is equal to the advantage of $\mathcal{R}$ in the DDH game, which is $\leq \mathsf{Adv}_{\mathsf{DDH}}$.

$$\Delta(\mathsf{G}_6, \mathsf{G}_7) \leq \mathsf{Adv}_{\mathsf{DDH}}$$

Game G_8 (SK is random unless responding to successful dishonest clients): G_8 is G_7 with the following changes:

- In S, if $\mathbb{w}_{\mathsf{C}} \neq \perp$, then parse $\mathbb{w}_{\mathsf{C}} \to (\mathsf{hpw}, s_{\mathsf{C}}, \cdot)$. If $F = f^*_{\mathsf{Vand}}(\mathsf{sid}_{\mathsf{S}}, \mathsf{hpw}, s_{\mathsf{C}} + s_{\mathsf{S}})$, then proceed as before.
- Otherwise (i.e. $\mathbb{w}_{\mathsf{C}} = \perp$ or $F \neq f^*_{\mathsf{Vand}}(\mathsf{sid}_{\mathsf{S}}, \mathsf{hpw}, s_{\mathsf{C}} + s_{\mathsf{S}})$), do not compute $d_K := \mathsf{H}_{\mathsf{d}}^{\mathsf{sid}_{\mathsf{S}}}(K)$ and $\mathsf{SK} := \mathsf{H}_{\mathsf{SK}}^{\mathsf{sid}_{\mathsf{S}}}(\mathsf{ssid}_{\mathsf{S}}, K)$ as normal. Instead sample $d_K \leftarrow_{\$} \mathbb{Z}_p$ and $\mathsf{SK} \leftarrow_{\$} \{0,1\}^{\kappa}$. Also, record $(\mathsf{ssid}_{\mathsf{S}}, \mathsf{SK})$ in association with proof π_{S}, and omit K from the truncated witness $\mathbb{w}_{\mathsf{S}}$ (see G_5). Finally, add $(\mathsf{sid}_{\mathsf{S}}, K)$ to set W.
- In C, if $\mathbb{w}_{\mathsf{S}}$ is a truncated (i.e. simulated) witness (i.e. π_{S} was generated by an honest S, see G_5) and $F = f^*_{\mathsf{Vand}}(\mathsf{sid}_{\mathsf{C}}, \mathsf{hpw}, s_{\mathsf{C}} + s_{\mathsf{S}})$ (see G_3), then retrieve the $(\mathsf{ssid}_{\mathsf{S}}, \mathsf{SK})$ recorded for π_{S}. Instead of computing $\mathsf{SK}' := \mathsf{H}_{\mathsf{SK}}^{\mathsf{sid}_{\mathsf{C}}}(\mathsf{ssid}_{\mathsf{C}}, K)$, if $\mathsf{ssid}_{\mathsf{C}} = \mathsf{ssid}_{\mathsf{S}}$ then set $\mathsf{SK}' := \mathsf{SK}$. Otherwise, sample $\mathsf{SK} \leftarrow_{\$} \{0,1\}^{\kappa}$.

Intuitively, the ideal adversary should learn nothing about the SK sampled by S unless S is responding to a dishonest client who supplies a successful guess of $\mathsf{hpw}, s_{\mathsf{C}}$. In this game change, we consider the cases that S is either responding to an honest client or to an unsuccessful dishonest client. In those cases, S in G_8 samples K and commits to it in C_K, but does nothing else with K. In particular, SK is an independently random value of which no information whatsoever is leaked to the ideal adversary.

Define KQueried to be the event that $\mathcal{Z}$ queries $\mathsf{H}_{\mathsf{d}}^{\mathsf{sid}_{\mathsf{S}}}(K)$ or $\mathsf{H}_{\mathsf{SK}}^{\mathsf{sid}_{\mathsf{S}}}(\mathsf{ssid}, K)$ (with any ssid) for any $(\mathsf{sid}_{\mathsf{S}}, K) \in W$. Unless KQueried occurs, the views of $\mathcal{Z}$ in G_7 and G_8 are identical, because $\mathcal{Z}$ never observes that d_K and SK were not derived from H_{d} and H_{SK}. Thus, $\Delta(\mathsf{G}_7, \mathsf{G}_8) \leq \Pr[\text{KQueried}]$. Furthermore, the only information $\mathcal{Z}$ sees about K is the commitment C_K, so the hiding property of ElGamal commitment should imply that it is hard for $\mathcal{Z}$ to compute K and trigger KQueried. Note that $\Pr[\text{KQueried}]$ is equal in G_7 and G_8, because the two games are identical up until the time KQueried occurs.

Given an environment $\mathcal{Z}$ that can trigger KQueried, we construct a reduction $\mathcal{R}$ that can solve DDH. The input to $\mathcal{R}$ is tuple $(A, B, C) \in \mathbb{G}^3$. $\mathcal{R}$ emulates game G_8 to $\mathcal{Z}$ with the following modifications:

- In S, if $\mathbb{w}_{\mathsf{C}} \neq \perp$, then $\mathcal{R}$ parses $\mathbb{w}_{\mathsf{C}} \to (\mathsf{hpw}, s_{\mathsf{C}}, \cdot)$. If $F = f^*_{\mathsf{Vand}}(\mathsf{sid}_{\mathsf{S}}, \mathsf{hpw}, s_{\mathsf{C}} + s_{\mathsf{S}})$, then $\mathcal{R}$ proceeds as before.
- Otherwise (i.e. $\mathbb{w}_{\mathsf{C}} = \perp$ or $F \neq f^*_{\mathsf{Vand}}(\mathsf{sid}_{\mathsf{S}}, \mathsf{hpw}, s_{\mathsf{C}} + s_{\mathsf{S}})$), instead of sampling $K \leftarrow_{\$} \mathbb{G}$, $C_K := \mathsf{ElG.Commit}(K; d_K)$, $\mathcal{R}$ samples $\delta, \mu, \nu \leftarrow_{\$} \mathbb{Z}_p$, $c_3 \leftarrow_{\$} \mathbb{G}$, and sets $C_K := (A \cdot g^{\delta}, g^{\mu} \cdot B^{\nu}, c_3)$. Also, $\mathcal{R}$ computes $K^* := c_3/(A^{\mu} \cdot C^{\nu}) \cdot (g^{\mu} \cdot B^{\nu})^{\delta}$ and adds $(\mathsf{sid}_{\mathsf{S}}, K^*)$ to set W.

Reduction $\mathcal{R}$ outputs 1 iff $\mathcal{Z}$ ever performs a query $\mathsf{H}_{\mathsf{d}}^{\mathsf{sid_S}}(K^*)$ or $\mathsf{H}_{\mathsf{SK}}^{\mathsf{sid_S}}(\mathsf{ssid}, K^*)$ (with any ssid) for any $(\mathsf{sid_S}, K^*) \in W$.

The views of $\mathcal{Z}$ in $\mathcal{R}$ and G_8 are identical. In both games, each C_K is a commitment to a random K that is never otherwise used (referring only to the cases of responding to an honest client or to an unsuccessful dishonest client). In G_8, the game samples these K values and commits to them outright. In $\mathcal{R}$ the reduction uses (A, B, C) to create the commitments (again applying DDH randomized self-reduction).

If (A, B, C) is a Diffie-Hellman tuple, then the committed K is K^*, so the KQUERIED event is exactly the same as the condition for $\mathcal{R}$ to output 1. Thus, $\Pr[1 \leftarrow \mathcal{R} \,|\, (A, B, C) \in DH] = \Pr[\text{KQUERIED}]$. If (A, B, C) is not a Diffie-Hellman tuple, the committed K and K^* are independently random, and no information about K^* is exposed to $\mathcal{Z}$. At most q_{S} $(\mathsf{sid_S}, K^*)$ tuples are recorded, so $\Pr[1 \leftarrow \mathcal{R} \,|\, (A, B, C) \notin DH] \leq (q_{\mathsf{S}} q_{\mathsf{H}})/p$. The absolute difference between these two probabilities is $\leq \mathsf{Adv}_{\mathsf{DDH}}$.

$$\Delta(\mathsf{G}_7, \mathsf{G}_8) \leq \mathsf{Adv}_{\mathsf{DDH}} + \frac{q_{\mathsf{S}} q_{\mathsf{H}}}{p}$$

Game G_9 (H_{SK} is programmed with SK when responding to successful dishonest clients): G_9 is G_8 with the following changes to S:

- If $\mathbb{w}_{\mathsf{C}} = \bot$, then proceed as before.
- Otherwise, parse $\mathbb{w}_{\mathsf{C}} \to (\mathsf{hpw}, s_{\mathsf{C}}, \cdot)$. If $F \neq f^*_{\mathsf{Vand}}(\mathsf{sid_S}, \mathsf{hpw}, s_{\mathsf{C}} + s_{\mathsf{S}})$, then still proceed as before.
- Otherwise, do not compute C_K, V'_0, and V'_1 as normal. Instead, sample $\mathsf{SK} \leftarrow_{\$} \{0,1\}^{\kappa}$, $K'_0, K'_1 \leftarrow_{\$} \mathbb{G}$, and $\beta \leftarrow_{\$} \{0,1\}$. Then, compute $V'_0 \leftarrow \mathsf{EIG.Enc}_{\mathsf{pk_C}}(K'_0)$, $V'_1 \leftarrow \mathsf{EIG.Enc}_{\mathsf{pk_C}}(K'_1)$, and $C_K \leftarrow \mathsf{EIG.Commit}(K'_\beta; \mathsf{H}_{\mathsf{d}}^{\mathsf{sid_S}}(K'_\beta))$. Finally, program $\mathsf{H}_{\mathsf{SK}}^{\mathsf{sid_S}}(\mathsf{ssid_S}, K'_\beta) := \mathsf{SK}$.

The change from G_8 to G_9 is mostly conceptual and merely shifts the order in which values are sampled and computed. The only actual difference from the previous game is that SK is sampled independently of K and then programmed into H_{SK}. Consequently, the views of $\mathcal{Z}$ in G_8 and G_9 are identical unless $\mathcal{Z}$ queries $\mathsf{H}_{\mathsf{SK}}^{\mathsf{sid_S}}(\mathsf{ssid_S}, K'_\beta)$ before the game samples $K'_\beta \leftarrow_{\$} \mathbb{G}$ and programs it. Only one S session runs per $\mathsf{ssid_S}$, so only one H_{SK} point is programmed per $\mathsf{ssid_S}$, and the environment's probability of having guessed one of them is negligible.

$$\Delta(\mathsf{G}_8, \mathsf{G}_9) \leq \frac{q_{\mathsf{H}}}{p}$$

Game G_{10} (the simulated world): The change from G_9 to G_{10} is purely conceptual. All interactions between the simulator and the honest parties now match the code of $\mathcal{F}_{\mathsf{CKEM}}$, so one can imagine the monolithic simulator now cleanly split into the two components $\mathcal{F}_{\mathsf{CKEM}}$ and $\mathsf{SIM}_{\mathsf{CKEM\text{-}Vand}}$.

$$\Delta(\mathsf{G}_9, \mathsf{G}_{10}) = 0$$
$$\mathsf{Exp}_{\mathcal{Z}}[\mathsf{G}_{10}] = \mathbf{Ideal}_{\mathcal{Z}}[\mathcal{F}_{\mathsf{CKEM}}, \mathsf{SIM}_{\mathsf{CKEM\text{-}Vand}}]$$

Summing up the distinguishing advantage bounds for each incremental game change yields an overall bound on $\mathcal{Z}$'s distinguishing advantage between the real and simulated worlds.

$$\Delta(\mathsf{G}_0, \mathsf{G}_{10}) \leq 3 \cdot \mathsf{Adv}_{\mathsf{DDH}} + \frac{{q_\mathsf{C}}^2 + {q_\mathsf{S}}^2 + q_\mathsf{S} q_\mathsf{H} + q_\mathsf{H}}{p}$$

$p = |\mathbb{G}|$ is an exponential function of the security parameter. If $\mathcal{Z}$ is efficient, then q_C, q_S, and q_H are polynomial functions of the security parameter. Furthermore, $\mathsf{Adv}_{\mathsf{DDH}}$ is a negligible function of the security parameter under the DDH assumption. Thus, the distinguishing advantage of any efficient $\mathcal{Z}$ between the real and simulated worlds is negligible.

Protocol $\mathsf{CKEM}_{\mathsf{Vand}}$ realizes $\mathcal{F}_{\mathsf{CKEM}}[\mathsf{SSEM}[f^*_{\mathsf{Vand}}, \mathsf{EIG}], \mathsf{SSEM}_{\mathsf{Adv}}[f^*_{\mathsf{Vand}}]]$.

H Proof of $\mathsf{CKEM}_{\mathsf{PoNR}}$ Security

In this section we prove Theorem 9, i.e. that protocol $\mathsf{CKEM}_{\mathsf{PoNR}}$ realizes $\mathcal{F}_{\mathsf{CKEM}}[\mathsf{SSEM}[f^*_{\mathsf{PoNR}}, \mathsf{EIG}], \mathsf{SSEM}_{\mathsf{Adv}}[f^*_{\mathsf{PoNR}}]]$.

Specifically, for any efficient adversary $\mathcal{A}^*$ against protocol $\mathsf{CKEM}_{\mathsf{PoNR}}$, there exists a simulator SIM such that no efficient environment $\mathcal{Z}$ can distinguish the view of $\mathcal{A}^*$ interacting with the real $\mathsf{CKEM}_{\mathsf{PoNR}}$ protocol and the view of SIM interacting with the ideal functionality $\mathcal{F}_{\mathsf{CKEM}}[\mathsf{SSEM}[f^*_{\mathsf{PoNR}}, \mathsf{EIG}], \mathsf{SSEM}_{\mathsf{Adv}}[f^*_{\mathsf{PoNR}}]]$ with advantage better than $3 \cdot \mathsf{Adv}_{\mathsf{DDH}} + \mathsf{Adv}_{\mathsf{DL}} + \frac{{q_\mathsf{C}}^2 + {q_\mathsf{S}}^2 + q_\mathsf{S} q_\mathsf{H} + q_\mathsf{H}}{p}$ where there are at most q_C client sessions, there are at most q_S server sessions, there are at most q_H queries to H_d and H_{SK}, and $\mathsf{Adv}_{\mathsf{DDH}}$ (resp. $\mathsf{Adv}_{\mathsf{DL}}$) is a bound on the DDH (resp. DL) advantage of any efficient algorithm.

Proof. For any adversary $\mathcal{A}^*$, we construct simulator $\mathsf{SIM}_{\mathsf{CKEM\text{-}PoNR}}$ as shown in Figures 16 and 17. Without loss of generality, we assume that $\mathcal{A}^*$ is a "dummy" adversary that merely passes messages to and from the environment $\mathcal{Z}$.

We now show that, for any efficient (i.e. PPT) $\mathcal{Z}$, the distinguishing advantage of $\mathcal{Z}$ between the real and simulated worlds is negligible. We denote the probability that $\mathcal{Z}$ outputs 1 in the interaction with the real world game or the simulated game as resp. $\mathbf{Real}_{\mathcal{Z}}[\mathsf{CKEM}_{\mathsf{PoNR}}]$ and $\mathbf{Ideal}_{\mathcal{Z}}[\mathcal{F}_{\mathsf{CKEM}}, \mathsf{SIM}_{\mathsf{CKEM\text{-}PoNR}}]$. We denote the probability that $\mathcal{Z}$ outputs 1 in an interaction with security game G_i as $\mathsf{Exp}_{\mathcal{Z}}[\mathsf{G}_i] = \Pr[1 \leftarrow \mathcal{Z}^{\mathsf{G}_i}(1^\kappa)]$, and we define $\Delta(\mathsf{G}_i, \mathsf{G}_j) = |\mathsf{Exp}_{\mathcal{Z}}[\mathsf{G}_i] - \mathsf{Exp}_{\mathcal{Z}}[\mathsf{G}_j]|$. The argument proceeds by a series of game changes, starting with game G_0 s.t. $\mathsf{Exp}_{\mathcal{Z}}[\mathsf{G}_0] = \mathbf{Real}_{\mathcal{Z}}[\mathsf{CKEM}_{\mathsf{PoNR}}]$ and ends with game G_{12} s.t. $\mathsf{Exp}_{\mathcal{Z}}[\mathsf{G}_{12}] = \mathbf{Ideal}_{\mathcal{Z}}[\mathcal{F}_{\mathsf{CKEM}}, \mathsf{SIM}_{\mathsf{CKEM\text{-}PoNR}}]$. We must show that $\Delta(\mathsf{G}_0, \mathsf{G}_{12})$ is negligible.

Game G_0 (the real world): The distinguisher $\mathcal{Z}$ interacts with $\mathsf{CKEM}_{\mathsf{PoNR}}$ (Figure 8) in the role of the honest parties and in the role of the adversary. Functionalities $\mathcal{F}_{\mathsf{COT}}$ (Figure 9) and $\mathcal{F}_{\mathsf{NIZK}}$ (Figure 1) interact with $\mathsf{CKEM}_{\mathsf{PoNR}}$ and with $\mathcal{Z}$ via the adversarial interface. H_x, H_d, and H_{SK} are true random oracles.

$$\mathsf{Exp}_{\mathcal{Z}}[\mathsf{G}_0] = \mathbf{Real}_{\mathcal{Z}}[\mathsf{CKEM}_{\mathsf{PoNR}}]$$

Game G_1 (statements never collide): G_1 is G_0 with the following changes:

- Program $\mathsf{H_d}$ and $\mathsf{H_{SK}}$ outputs uniformly at random, except where special $\mathsf{H_{SK}}$ programming is noted below.
- On $(\textsc{Ceval}, \mathsf{ssid_C}, \mathsf{C}, (\mathsf{sid_C}, C_F))$ from $\mathcal{F}_{\mathsf{CKEM}}$:
  - Sample $\mathsf{pk_C} \leftarrow_\$ \mathbb{G}$, $\boldsymbol{G} \leftarrow_\$ (\mathbb{G}^2)^N$, and $\hat{G} \leftarrow_\$ \mathbb{G}^2$.
  - Sample $\boldsymbol{m} \leftarrow_\$ ((\mathbb{Z}_p^N)^t)^2$ and $\boldsymbol{d}_m \leftarrow_\$ ((\mathbb{Z}_p)^t)^2$. For each $j \in [t]$, compute $C_{m_{j,0}} \leftarrow \mathsf{Ped}^N((m_{i,j,0})_{i\in[N]}; d_{m_{j,0}})$ and $C_{m_{j,1}} \leftarrow \mathsf{Ped}^N((m_{i,j,1})_{i\in[N]}; d_{m_{j,1}})$.[†] Define $\boldsymbol{C}_m \leftarrow (C_{m_{j,0}}, C_{m_{j,1}})_{j\in[t]}$.
  - Send $(\textsc{Prove}, \mathsf{C}, \mathsf{ssid_C}, (\mathsf{sid_C}, \mathsf{pk_C}, \boldsymbol{G}, \hat{G}, \boldsymbol{C}_m))$ to $\mathcal{A}^*$ as if from $\mathcal{F}_{\mathsf{NIZK}}[\mathcal{R}_\mathsf{C}]$, await response $(\textsc{Proof}, \mathsf{ssid_C}, \pi_\mathsf{C})$, and save record $(\mathcal{R}_\mathsf{C}, (\mathsf{sid_C}, \mathsf{pk_C}, \boldsymbol{G}, \hat{G}, \boldsymbol{C}_C), \bot, \pi_\mathsf{C}, \mathsf{ssid_C})$.
  - Send $(\mathsf{sid_C}, \mathsf{pk_C}, \boldsymbol{G}, \hat{G}, \pi_\mathsf{C})$ to $\mathcal{A}^*$ as if from C to S.
  - For each $j \in [t]$, save record $(\textsc{Send}, (\mathsf{ssid_C}|j), (m_{i,j,0})_{i\in[N]}, d_{m_{j,0}}, (m_{i,j,1})_{i\in[N]}, d_{m_{j,1}})$ marked fresh and send $(\textsc{Send}, (\mathsf{ssid_C}|j), \mathsf{C}, C_{m_{j,0}}, C_{m_{j,1}})$ to $\mathcal{A}^*$ as if from $\mathcal{F}_{\mathsf{COT}}$.
  - Await (C_K, V, π_S) from $\mathcal{A}^*$ to C.
  - Define $\mathbb{x}_\mathsf{S} := (C_F, \mathsf{pk_C}, \boldsymbol{G}, \hat{G}, \boldsymbol{C}_m, C_K, V)$. If there exists no record $(\mathcal{R}_\mathsf{S}, \mathbb{x}_\mathsf{S}, \mathbb{w}_\mathsf{S}, \pi_\mathsf{S}, \mathsf{ssid_S})$ then send $(\textsc{Verify}, \mathsf{C}, \mathsf{ssid_C}, \mathbb{x}_\mathsf{S}, \pi_\mathsf{S})$ to $\mathcal{A}^*$ as if from $\mathcal{F}_{\mathsf{NIZK}}[\mathcal{R}_\mathsf{S}]$, await response $(\textsc{Witness}, \mathsf{ssid_C}, \mathbb{w}_\mathsf{S})$, and save record $(\mathcal{R}_\mathsf{S}, \mathbb{x}_\mathsf{S}, \mathbb{w}_\mathsf{S}, \pi_\mathsf{S}, \bot)$.
  - If $\mathbb{w}_\mathsf{S} \neq \bot$ then parse $\mathbb{w}_\mathsf{S} \to (F, d_F, s_\mathsf{S}, \cdot, \cdot, \cdot, K, d_K, \cdot)$, and...
    - If $(\mathbb{x}_\mathsf{S}, \mathbb{w}_\mathsf{S}) \notin \mathcal{R}_\mathsf{S} \vee \mathsf{H_d^{sid_C}}(K) \neq d_K$: Send $(\textsc{AdvSeval}, \mathsf{ssid_C}, \bot, 0^\kappa)$ to $\mathcal{F}_{\mathsf{CKEM}}$.
    - Else: Send $(\textsc{AdvSeval}, \mathsf{ssid_C}, (\mathsf{sid_C}, F, s_\mathsf{S}, d_F), \mathsf{H_{SK}^{sid_C}}(\mathsf{ssid_C}, K))$ to $\mathcal{F}_{\mathsf{CKEM}}$.
  - If $\mathbb{w}_\mathsf{S} = \bot$, then...
    - If $\mathsf{ssid_C} \neq \mathsf{ssid_S}$: Send $(\textsc{AdvSeval}, \mathsf{ssid_C}, \bot, 0^\kappa)$ to $\mathcal{F}_{\mathsf{CKEM}}$.
    - Else: Send $(\textsc{Cout}, \mathsf{ssid_C})$ to $\mathcal{F}_{\mathsf{CKEM}}$.
- On $(\textsc{Seval}, \mathsf{ssid_S}, \mathsf{S}, (\mathsf{sid_S}, C_F))$ from $\mathcal{F}_{\mathsf{CKEM}}$:
  - Await $(\mathsf{sid_C}, \mathsf{pk_C}, \boldsymbol{G}, \hat{G}, \pi_\mathsf{C})$ from $\mathcal{A}^*$ to S.
  - For each $j \in [t]$, send $(\textsc{Rcv}, (\mathsf{ssid_S}|j), \mathsf{S})$ to $\mathcal{A}^*$ as if from $\mathcal{F}_{\mathsf{COT}}$.
  - For each $j \in [t]$, await $(\textsc{Complete}, \mathsf{ssid}'_j, (\mathsf{ssid_S}|j))$ from $\mathcal{A}^*$ to $\mathcal{F}_{\mathsf{COT}}$ where there exists record $(\textsc{Send}, \mathsf{ssid}'_j, (m_{i,j,0})_{i\in[N]}, d_{m_{j,0}}, (m_{i,j,1})_{i\in[N]}, d_{m_{j,1}})$ marked fresh. Mark it $\mathsf{completed}$.
  - Define $\boldsymbol{C}_m \leftarrow (\mathsf{Ped}^N((m_{i,j,0})_{i\in[N]}; d_{m_{j,0}}), \mathsf{Ped}^N((m_{i,j,1})_{i\in[N]}; d_{m_{j,1}}))_{j\in[t]}$.
  - Define $\mathbb{x}_\mathsf{C} := (\mathsf{sid_S}, \mathsf{pk_C}, \boldsymbol{G}, \hat{G}, \boldsymbol{C}_m)$. If there exists no record $(\mathcal{R}_\mathsf{C}, \mathbb{x}_\mathsf{C}, \mathbb{w}_\mathsf{C}, \pi_\mathsf{C}, \mathsf{ssid_C})$ then send $(\textsc{Verify}, \mathsf{S}, \mathsf{ssid_S}, \mathbb{x}_\mathsf{C}, \pi_\mathsf{C})$ to $\mathcal{A}^*$ as if from $\mathcal{F}_{\mathsf{NIZK}}[\mathcal{R}_\mathsf{C}]$, await response $(\textsc{Witness}, \mathsf{ssid_S}, \mathbb{w}_\mathsf{C})$, and save record $(\mathcal{R}_\mathsf{C}, \mathbb{x}_\mathsf{C}, \mathbb{w}_\mathsf{C}, \pi_\mathsf{C}, \bot)$.[†]
  - If $\mathsf{sid_C} \neq \mathsf{sid_S} \vee (\mathbb{w}_\mathsf{C} \neq \bot \wedge (\mathbb{x}_\mathsf{C}, \mathbb{w}_\mathsf{C}) \notin \mathcal{R}_\mathsf{C})$: Send $(\textsc{AdvCeval}, \mathsf{ssid_S}, \bot)$ to $\mathcal{F}_{\mathsf{CKEM}}$, receive response $(\textsc{Cout}, \mathsf{ssid_S}, \bot)$, and break (i.e. don't run the following code).
  - If $\mathbb{w}_\mathsf{C} \neq \bot$: Parse $\mathbb{w}_\mathsf{C} \to (\mathsf{hpw}, s_\mathsf{C}, \cdot, \cdot, \cdot, \cdot)$, send $(\textsc{AdvCeval}, \mathsf{ssid_S}, (\mathsf{sid_S}, \mathsf{hpw}, s_\mathsf{C}))$ to $\mathcal{F}_{\mathsf{CKEM}}$, and receive response $(\textsc{Cout}, \mathsf{ssid_S}, \mathsf{SK})$.
  - If $\mathbb{w}_\mathsf{C} = \bot$ then define $\mathsf{SK} := \bot$, and...
    - If $\mathsf{ssid_S} \neq \mathsf{ssid_C}$: Send $(\textsc{AdvCeval}, \mathsf{ssid_S}, \bot)$ to $\mathcal{F}_{\mathsf{CKEM}}$ and receive response $(\textsc{Cout}, \mathsf{ssid_S}, \bot)$.
    - Else: Send $(\textsc{Sout}, \mathsf{ssid_S})$ to $\mathcal{F}_{\mathsf{CKEM}}$.
  - Sample $K' \leftarrow_\$ \mathbb{G}$, and compute $V \leftarrow \mathsf{EIG.Enc_{pk_C}}(K')$.
  - If $\mathsf{SK} \neq \bot$ then compute $C_K \leftarrow \mathsf{EIG.Commit}(K'; \mathsf{H_d^{sid_S}}(K'))$, and program $\mathsf{H_{SK}^{sid_S}}(\mathsf{ssid_S}, K') := \mathsf{SK}$. Else $(\mathsf{SK} = \bot)$ sample $C_K \leftarrow_\$ \mathbb{G}^3$.
  - Define $\mathbb{x}_\mathsf{S} := (C_F, \mathsf{pk_C}, \boldsymbol{G}, \hat{G}, \boldsymbol{C}_m, C_K, V)$. Send $(\textsc{Prove}, \mathsf{S}, \mathsf{ssid_S}, \mathbb{x}_\mathsf{S})$ to $\mathcal{A}^*$ as if from $\mathcal{F}_{\mathsf{NIZK}}[\mathcal{R}_\mathsf{S}]$, await response $(\textsc{Proof}, \mathsf{ssid_S}, \pi_\mathsf{S})$, and save record $(\mathcal{R}_\mathsf{S}, \mathbb{x}_\mathsf{S}, \bot, \pi_\mathsf{S}, \mathsf{ssid_S})$.
  - Send (C_K, V, π_S) to $\mathcal{A}^*$ as if from S to C.

Fig. 16. $\mathsf{SIM_{CKEM\text{-}PoNR}}$: Simulator for protocol $\mathsf{CKEM_{PoNR}}$, part 1: Honest C and S

– On $(\textsc{Send}, \mathsf{ssid}, m_0, d_0, m_1, d_1)$ from $\mathcal{A}^*$ to $\mathcal{F}_{\mathsf{COT}}$:[†]
  - Save record $(\textsc{Send}, \mathsf{ssid}, m_0, d_0, m_1, d_1)$ marked fresh.
  - Send $(\textsc{Send}, \mathsf{ssid}, \mathcal{A}^*, \mathsf{Ped}^N(m_0; d_0), \mathsf{Ped}^N(m_1; d_1))$ to $\mathcal{A}^*$ as if from $\mathcal{F}_{\mathsf{COT}}$.
– On $(\textsc{Rcv}, \mathsf{ssid}, \sigma)$ from $\mathcal{A}^*$ to $\mathcal{F}_{\mathsf{COT}}$:
  - Send $(\textsc{Rcv}, \mathsf{ssid}, \mathcal{A}^*)$ to $\mathcal{A}^*$ as if from $\mathcal{F}_{\mathsf{COT}}$.
  - Await $(\textsc{Complete}, \mathsf{ssid}', \mathsf{ssid})$ from $\mathcal{A}^*$ to $\mathcal{F}_{\mathsf{COT}}$ where there exists record $(\textsc{Send}, \mathsf{ssid}', m_0, d_0, m_1, d_1)$ marked fresh. Mark it completed.
  - Send $(\textsc{Complete}, \mathsf{ssid}, m_\sigma, d_\sigma, \mathsf{Ped}^N(m_{1-\sigma}, d_{1-\sigma}))$ to $\mathcal{A}^*$ as if from $\mathcal{F}_{\mathsf{COT}}$.

If $\mathcal{Z}$ ever supplies two different openings $C = \mathsf{Ped}^N(m; d) = \mathsf{Ped}^N(m^*; d^*), m \neq m^*$ to a Ped^N commitment, immediately abort the game. There are three particular avenues by which these commitment openings could arise, and they are marked by symbol [†] in the simulator. First, honest clients produce and output Ped^N commitments. Second, $\mathcal{Z}$ supplies commitment openings to $\mathcal{F}_{\mathsf{COT}}$ when acting as a COT sender. Third, $\mathcal{Z}$ supplies commitment openings in $\mathcal{R}_{\mathsf{C}}$ witnesses, each of which includes (for every $j \in [t], \sigma \in \{0, 1\}$) a decommitment $d_{m_{j,\sigma}}$, as well as $s_{\mathsf{C}}[j], (\delta_{i,j})_{i\in[N]}$, which imply message $(m_{i,j,\sigma})_{i\in[N]}$.

Fig. 17. $\mathsf{SIM}_{\mathsf{CKEM\text{-}PoNR}}$: Simulator for protocol $\mathsf{CKEM}_{\mathsf{PoNR}}$, part 2: Adversarial $\mathcal{F}_{\mathsf{COT}}$ interactions

– If two C sessions ever pick the same pk_{C}, the game immediately aborts.
– If two S sessions ever pick the same K, the game immediately aborts.

Per the code of $\mathcal{F}_{\mathsf{NIZK}}$, it is possible for a single statement-proof pair $(\mathbb{x}, \pi)$ to correspond to more than one witness $\mathbb{w}$ iff statement $\mathbb{x}$ is proven more than once by honest parties. This game change ensures that the $\mathbb{x}_{\mathsf{C}}$ and $\mathbb{x}_{\mathsf{S}}$ statements proven by honest parties never repeat. Therefore, in future games, we can retrieve the one $\mathbb{w}$ for any $(\mathbb{x}, \pi)$ without ambiguity.

Every pk_{C} (resp. K) is sampled uniformly at random from $\mathbb{G}$. Therefore, the probability of a collision is upper-bounded by ${q_{\mathsf{C}}}^2/p$ (resp. ${q_{\mathsf{S}}}^2/p$). If no collision occurs, G_0 and G_1 are identical.

$$\Delta(\mathsf{G}_0, \mathsf{G}_1) \leq \frac{{q_{\mathsf{C}}}^2 + {q_{\mathsf{S}}}^2}{p}$$

Game G_2 (client NIZKs are simulated): G_2 is G_1 with only one change: $\mathbb{w}_{\mathsf{C}}$ is omitted when C calls $\mathcal{F}_{\mathsf{NIZK}}[\mathcal{R}_{\mathsf{C}}].\textsc{Prove}$. Since honest clients never attempt to prove false statements, the check for $(\mathbb{x}, \mathbb{w}) \in \mathcal{R}_{\mathsf{C}}$ is simply skipped, and instead of a witness a special symbol $\mathbb{w} := \perp$ is recorded to represent that the produced proof is correct. The code of $\mathcal{F}_{\mathsf{NIZK}}[\mathcal{R}_{\mathsf{C}}].\textsc{Verify}$ is modified to always accept proofs for records with $\mathbb{w} = \perp$.

The views of $\mathcal{Z}$ in G_1 and G_2 are identical.

$$\Delta(\mathsf{G}_1, \mathsf{G}_2) = 0$$

Game G_3 (clients do not decrypt V): G_3 is G_2 with the following changes to C:

– Instead of sampling $(\mathsf{sk}_\mathsf{C}, \mathsf{pk}_\mathsf{C}) \leftarrow \mathsf{ElG.KG}$, only sample $\mathsf{pk}_\mathsf{C} \leftarrow_\$ \mathbb{G}$.
– After verifying π_S, do not decrypt V. Instead, parse $\mathbb{w}_\mathsf{S} \rightarrow (F, \cdot, s_\mathsf{S}, \cdot, \cdot, \cdot, K, d_K, \cdot)$. If $\mathsf{H}_\mathsf{d}^{\mathsf{sid}_\mathsf{C}}(K) = d_K \ \wedge \ F = f^*_{\mathsf{PoNR}}(\mathsf{sid}_\mathsf{C}, \mathsf{hpw}, s_\mathsf{C} \oplus s_\mathsf{S})$, then output $\mathsf{SK}' := \mathsf{H}_{\mathsf{SK}}^{\mathsf{sid}_\mathsf{C}}(\mathsf{ssid}_\mathsf{C}, K)$. Otherwise, output $\mathsf{SK}' \leftarrow_\$ \{0,1\}^\kappa$.

Notice that C's changed code only runs if π_S verifies. $\mathcal{R}_\mathsf{S}$ guarantees that the received V and C_K are consistent with correct server behavior on $\mathbb{w}_\mathsf{S}$, e.g. $C_K = \mathsf{ElG.Commit}(K; d_K)$. Therefore it is not necessary for C to refer to V and C_K directly. In the previous game, C will decrypt K from V iff $F = f^*_{\mathsf{PoNR}}(\mathsf{sid}_\mathsf{C}, \mathsf{hpw}, s_\mathsf{C} + s_\mathsf{S})$, and if additionally $\mathsf{H}_\mathsf{d}^{\mathsf{sid}_\mathsf{C}}(K) = d_K$ then C will use K to derive SK'. If either of those conditions does not occur, then C will output a random SK'. The behavior of C in this game is exactly the same.

The views of $\mathcal{Z}$ in G_2 and G_3 are identical.

$$\Delta(\mathsf{G}_2, \mathsf{G}_3) = 0$$

Game G_4 ($\boldsymbol{G}$ and $\hat{G}$ are random): G_4 is G_3 with only one change: instead of computing $(g_i)_{i\in[N]}$, $\boldsymbol{G} = (G_i)_{i\in[N]}$, $\hat{g}$, and $\hat{G}$ as in $\mathsf{CKEM}_{\mathsf{PoNR}}$, C simply samples $\boldsymbol{G} \leftarrow_\$ (\mathbb{G}^2)^N$ and $\hat{G} \leftarrow_\$ \mathbb{G}^2$.

Recall that client NIZKs are simulated, so it is not a problem that $\mathbb{x}_\mathsf{C}$ no longer has a correct witness. Additionally, the response V that S computes from $\boldsymbol{G}$ and $\hat{G}$ is not decrypted and used by C, so it is not a problem that these responses are formed from random G_i and $\hat{G}$ ciphertexts.

Given an environment $\mathcal{Z}$ that can distinguish G_3 from G_4, we construct a reduction $\mathcal{R}$ that can solve DDH. The input to $\mathcal{R}$ is tuple $(A, B, C) \in \mathbb{G}^3$. $\mathcal{R}$ emulates game G_3 to $\mathcal{Z}$ with the following modifications to the code of C:

– Instead of sampling $\mathsf{pk}_\mathsf{C} \leftarrow_\$ \mathbb{G}$, $\mathcal{R}$ samples $\delta \leftarrow_\$ \mathbb{Z}_p$ and sets $\mathsf{pk}_\mathsf{C} := A \cdot g^\delta$.
– Instead of encrypting by $d_{G_i} \leftarrow_\$ \mathbb{Z}_p$, $G_i \leftarrow \mathsf{ElG.Enc}_{\mathsf{pk}_\mathsf{C}}(g_i; d_{G_i})$, $\mathcal{R}$ samples $\mu, \nu \leftarrow_\$ \mathbb{Z}_p$ and sets $G_i := (g^\mu \cdot B^\nu, g_i \cdot (A^\mu \cdot C^\nu) \cdot (g^\mu \cdot B^\nu)^\delta)$.
– Instead of encrypting by $d_{\hat{G}} \leftarrow_\$ \mathbb{Z}_p$, $\hat{G} \leftarrow \mathsf{ElG.Enc}_{\mathsf{pk}_\mathsf{C}}(\hat{g}; d_{\hat{G}})$, $\mathcal{R}$ samples $\mu, \nu \leftarrow_\$ \mathbb{Z}_p$ and sets $\hat{G} := (g^\mu \cdot B^\nu, \hat{g} \cdot (A^\mu \cdot C^\nu) \cdot (g^\mu \cdot B^\nu)^\delta)$.

Here we apply the well-known DDH randomized self-reduction [44,39]. If (A, B, C) is a Diffie-Hellman tuple, then $(A, g^\mu \cdot B^\nu, A^\mu \cdot C^\nu)$ is a random Diffie-Hellman tuple. If not, then it is a uniformly random tuple. For every C session, we apply a "shift" of δ to get a freshly random pk_C. If (A, B, C) is a Diffie-Hellman tuple, then each G_i (resp. $\hat{G}$) ciphertext is a correct ElGamal encryption of g_i (resp. $\hat{g}$) under pk_C, so the view of $\mathcal{Z}$ is identical to that in G_3. If (A, B, C) is not a Diffie-Hellman tuple, then each G_i and $\hat{G}$ ciphertext is independently random, so the view of $\mathcal{Z}$ is identical to that in G_4.

Reduction $\mathcal{R}$ outputs 1 iff $\mathcal{Z}$ outputs 1. Thus, the advantage of $\mathcal{Z}$ at distinguishing between G_3 and G_4 is equal to the advantage of $\mathcal{R}$ in the DDH game, which is $\leq \mathsf{Adv}_{\mathsf{DDH}}$.

$$\Delta(\mathsf{G}_3, \mathsf{G}_4) \leq \mathsf{Adv}_{\mathsf{DDH}}$$

Game G_5 ($\boldsymbol{m}$ is random): G_5 is G_4 with only one change: instead of computing $m_{i,j,0}$ and $m_{i,j,1}$ (for all $i \in [N], j \in [t]$) as in $\mathsf{CKEM}_{\mathsf{PoNR}}$, C simply samples $\boldsymbol{m} \leftarrow_{\$} ((\mathbb{Z}_p^N)^t)^2$.

As in the previous game change, we recall that client NIZKs are simulated, so it is not a problem that $\mathbb{x}_{\mathsf{C}}$ no longer has a correct witness. Additionally, the response V that S computes from the received m values is not decrypted and used by C, so it is not a problem that these m values are random.

In this game, $\mathcal{Z}$ sees a Pedersen commitment to $(m_{i,j,0})_{i\in[N]}$ and $(m_{i,j,1})_{i\in[N]}$ for every $j \in [t]$. If playing the role of a dishonest server, $\mathcal{Z}$ then sees an opening for exactly one of the two commitments. Pedersen commitments (Section 2) are perfectly hiding, and for any one revealed $(m_{i,j,\sigma})_{i\in[N]}$ there exists $(\delta_{i,j})_{i\in[N]}$ that could have been chosen in G_4 to produce an identical view.

The views of $\mathcal{Z}$ in G_4 and G_5 are identical.

$$\Delta(\mathsf{G}_4, \mathsf{G}_5) = 0$$

Game G_6 ($\boldsymbol{C}_m$ commitments are perfectly binding): G_6 is G_5 with only one change: if $\mathcal{Z}$ ever supplies two different openings $C = \mathsf{Ped}^N(m; d) = \mathsf{Ped}^N(m^*; d^*), m \neq m^*$ to a Ped^N commitment, immediately abort the game. There are three particular avenues by which these commitment openings could arise. First, honest clients produce and output Ped^N commitments. Second, $\mathcal{Z}$ supplies commitment openings to $\mathcal{F}_{\mathsf{COT}}$ when acting as a COT sender. Third, $\mathcal{Z}$ supplies commitment openings in $\mathcal{R}_{\mathsf{C}}$ witnesses, each of which includes (for every $j \in [t], \sigma \in \{0, 1\}$) a decommitment $d_{m_{j,\sigma}}$, as well as $s_{\mathsf{C}}[j], (\delta_{i,j})_{i\in[N]}$, which imply message $(m_{i,j,\sigma})_{i\in[N]}$.

Future games will assume that the OT messages $\boldsymbol{m}^{\mathsf{S}}$ received by S are consistent with the witness $\mathbb{w}_{\mathsf{C}}$ of the corresponding client proof π_{C}. $\mathcal{F}_{\mathsf{NIZK}}$ guarantees the soundness of π_{C}, but if $\mathcal{Z}$ provides different openings of a commitment to $\mathcal{F}_{\mathsf{NIZK}}$ and $\mathcal{F}_{\mathsf{COT}}$, then these future games will not behave correctly.

Unless this binding-break abort occurs, the views of $\mathcal{Z}$ in G_5 and G_6 are identical. Given an environment $\mathcal{Z}$ that can trigger this abort, we construct a reduction $\mathcal{R}$ that can solve DL. $\mathcal{R}$ is the straightforward and well-known "DL relation" reduction (e.g. [35]) that embeds the DL challenge in the Ped^N bases $h_1, \ldots, h_N$. The probability of this reduction's success is $\leq \mathsf{Adv}_{\mathsf{DL}}$.

$$\Delta(\mathsf{G}_5, \mathsf{G}_6) \leq \mathsf{Adv}_{\mathsf{DL}}$$

Game G_7 (server NIZKs are simulated): G_7 is G_6 with the following changes:

- $\mathbb{w}_{\mathsf{S}}$ is truncated to $(F, \cdot, s_{\mathsf{S}}, \cdot, \cdot, \cdot, K, \cdot, \cdot)$ when S calls $\mathcal{F}_{\mathsf{NIZK}}[\mathcal{R}_{\mathsf{S}}]$.Prove. Since honest servers never attempt to prove false statements, the check for $(\mathbb{x}, \mathbb{w}) \in \mathcal{R}_{\mathsf{S}}$ is simply skipped, and the code of $\mathcal{F}_{\mathsf{NIZK}}[\mathcal{R}_{\mathsf{S}}]$.Verify is modified to always accept proofs with witnesses of this truncated form.
- In the previous game, C parses F, s_{S}, K, and d_K from $\mathbb{w}_{\mathsf{S}}$ (see G_3), but d_K is now missing from the recorded witnesses of honestly generated server proofs. However, d_K is only used to check whether $\mathsf{H}_{\mathsf{d}}^{\mathsf{sid}_{\mathsf{C}}}(K) = d_K$, and this

condition always holds for honest servers, so the code of C is modified to simply skip that check when handling a witness of this truncated form.

The views of $\mathcal{Z}$ in G_6 and G_7 are identical.

$$\Delta(\mathsf{G}_6, \mathsf{G}_7) = 0$$

Game G_8 (V is random when responding to unsuccessful dishonest clients): G_8 is G_7 with the following changes to S after verifying π_{C} and checking that $\mathsf{sid}_{\mathsf{C}} = \mathsf{sid}_{\mathsf{S}}$:

- If $\mathbb{w}_{\mathsf{C}} = \perp$ (i.e. π_{C} was generated by an honest C, see G_2), then proceed as before.
- Otherwise, parse $\mathbb{w}_{\mathsf{C}} \to (\mathsf{hpw}, s_{\mathsf{C}}, \cdot, \cdot, \cdot, \cdot)$. If $F = f^*_{\mathsf{PoNR}}(\mathsf{sid}_{\mathsf{S}}, \mathsf{hpw}, s_{\mathsf{C}} \oplus s_{\mathsf{S}})$, then still proceed as before.
- Otherwise, do not compute V as normal. Instead, simply sample $V \leftarrow_{\$} \mathbb{G}^2$.

Recall that server NIZKs are simulated, so it is not a problem that S no longer knows a witness for $\mathbb{x}_{\mathsf{S}}$. Furthermore, we are only modifying the behavior of S in the case that $\mathbb{w}_{\mathsf{C}} \neq \perp$, in which case $\boldsymbol{G}$, $\hat{G}$, and $\boldsymbol{m}^{\mathsf{S}}$ must be correctly formed for $\mathsf{sid}_{\mathsf{S}}, \mathsf{hpw}, s_{\mathsf{C}}$ (by $\mathcal{F}_{\mathsf{NIZK}}[\mathcal{R}_{\mathsf{C}}]$ soundness and $\boldsymbol{C}_m$ binding, see G_6). Therefore, the algebra underlying $\mathsf{CKEM}_{\mathsf{PoNR}}$ implies that, if $F \neq f^*_{\mathsf{PoNR}}(\mathsf{sid}_{\mathsf{S}}, \mathsf{hpw}, s_{\mathsf{C}} \oplus s_{\mathsf{S}})$, then V is a uniformly random ElGamal ciphertext.

Thus, the views of $\mathcal{Z}$ in G_7 and G_8 are identical.

$$\Delta(\mathsf{G}_7, \mathsf{G}_8) = 0$$

Game G_9 (V is random when responding to honest clients): G_9 is G_8 with only one change: if $\mathbb{w}_{\mathsf{C}} = \perp$, then S samples $V \leftarrow_{\$} \mathbb{G}^2$ instead of computing V as before.

Since S should be computing V under a public key pk_{C} whose secret key is unknown to $\mathcal{Z}$, it should not be possible for $\mathcal{Z}$ to distinguish the replacement of V with a random ciphertext.

Given an environment $\mathcal{Z}$ that can distinguish G_8 from G_9, we construct a reduction $\mathcal{R}$ that can solve DDH. The input to $\mathcal{R}$ is tuple $(A, B, C) \in \mathbb{G}^3$. $\mathcal{R}$ emulates game G_8 to $\mathcal{Z}$ with the following modifications:

- In C, instead of sampling $\mathsf{pk}_{\mathsf{C}} \leftarrow_{\$} \mathbb{G}$, $\mathcal{R}$ samples $\delta \leftarrow_{\$} \mathbb{Z}_p$ and sets $\mathsf{pk}_{\mathsf{C}} := A \cdot g^{\delta}$. δ is recorded in association with proof π_{C}.
- In S, instead of encrypting by $d_V \leftarrow_{\$} \mathbb{Z}_p$, $V \leftarrow \mathsf{ElG.Enc}_{\mathsf{pk}_{\mathsf{C}}}(K; d_V) \cdot (\cdots)$, $\mathcal{R}$ samples $\mu, \nu \leftarrow_{\$} \mathbb{Z}_p$ and sets $V := (g^{\mu} \cdot B^{\nu}, K \cdot (A^{\mu} \cdot C^{\nu}) \cdot (g^{\mu} \cdot B^{\nu})^{\delta}) \cdot (\cdots)$, where δ is as recorded for π_{C}.

Again we utilize DDH randomized self-reduction with a random δ "shift" applied to each pk_{C}. If (A, B, C) is a Diffie-Hellman tuple, then ciphertext V is formed exactly as in $\mathsf{CKEM}_{\mathsf{PoNR}}$ and the view of $\mathcal{Z}$ is identical to that in G_8. If (A, B, C) is not a Diffie-Hellman tuple, then ciphertext V is independently random and the view of $\mathcal{Z}$ is identical to that in G_9.

Reduction $\mathcal{R}$ outputs 1 iff $\mathcal{Z}$ outputs 1. Thus the advantage of $\mathcal{Z}$ at distinguishing between G_8 and G_9 is equal to the advantage of $\mathcal{R}$ in the DDH game, which is $\leq \mathsf{Adv}_{\mathsf{DDH}}$.

$$\Delta(\mathsf{G}_8, \mathsf{G}_9) \leq \mathsf{Adv}_{\mathsf{DDH}}$$

Game G_{10} (SK is random unless responding to successful dishonest clients): G_{10} is G_9 with the following changes:

- In S, if $\mathbb{w}_{\mathsf{C}} \neq \perp$, then parse $\mathbb{w}_{\mathsf{C}} \to (\mathsf{hpw}, s_{\mathsf{C}}, \cdot, \cdot, \cdot, \cdot)$. If $F = f^*_{\mathsf{PoNR}}(\mathsf{sid}_{\mathsf{S}}, \mathsf{hpw}, s_{\mathsf{C}} \oplus s_{\mathsf{S}})$, then proceed as before.
- Otherwise (i.e. $\mathbb{w}_{\mathsf{C}} = \perp$ or $F \neq f^*_{\mathsf{PoNR}}(\mathsf{sid}_{\mathsf{S}}, \mathsf{hpw}, s_{\mathsf{C}} \oplus s_{\mathsf{S}})$), do not compute $d_K := \mathsf{H}_{\mathsf{d}}^{\mathsf{sid}_{\mathsf{S}}}(K)$ and $\mathsf{SK} := \mathsf{H}_{\mathsf{SK}}^{\mathsf{sid}_{\mathsf{S}}}(\mathsf{ssid}_{\mathsf{S}}, K)$ as normal. Instead sample $d_K \leftarrow_{\$} \mathbb{Z}_p$ and $\mathsf{SK} \leftarrow_{\$} \{0,1\}^{\kappa}$. Also, record $(\mathsf{ssid}_{\mathsf{S}}, \mathsf{SK})$ in association with proof π_{S}, and omit K from the truncated witness $\mathbb{w}_{\mathsf{S}}$ (see G_7). Finally, add $(\mathsf{sid}_{\mathsf{S}}, K)$ to set W.
- In C, if $\mathbb{w}_{\mathsf{S}}$ is a truncated (i.e. simulated) witness (i.e. π_{S} was generated by an honest S, see G_7) and $F = f^*_{\mathsf{PoNR}}(\mathsf{sid}_{\mathsf{C}}, \mathsf{hpw}, s_{\mathsf{C}} \oplus s_{\mathsf{S}})$ (see G_3), then retrieve the $(\mathsf{ssid}_{\mathsf{S}}, \mathsf{SK})$ recorded for π_{S}. Instead of computing $\mathsf{SK}' := \mathsf{H}_{\mathsf{SK}}^{\mathsf{sid}_{\mathsf{C}}}(\mathsf{ssid}_{\mathsf{C}}, K)$, if $\mathsf{ssid}_{\mathsf{C}} = \mathsf{ssid}_{\mathsf{S}}$ then set $\mathsf{SK}' := \mathsf{SK}$. Otherwise, sample $\mathsf{SK} \leftarrow_{\$} \{0,1\}^{\kappa}$.

Intuitively, the ideal adversary should learn nothing about the SK sampled by S unless S is responding to a dishonest client who supplies a successful guess of $\mathsf{hpw}, s_{\mathsf{C}}$. In this game change, we consider the cases that S is either responding to an honest client or to an unsuccessful dishonest client. In those cases, S in G_{10} samples K and commits to it in C_K, but does nothing else with K. In particular, SK is an independently random value of which no information whatsoever is leaked to the ideal adversary.

Define KQueried to be the event that $\mathcal{Z}$ queries $\mathsf{H}_{\mathsf{d}}^{\mathsf{sid}_{\mathsf{S}}}(K)$ or $\mathsf{H}_{\mathsf{SK}}^{\mathsf{sid}_{\mathsf{S}}}(\mathsf{ssid}, K)$ (with any ssid) for any $(\mathsf{sid}_{\mathsf{S}}, K) \in W$. Unless KQueried occurs, the views of $\mathcal{Z}$ in G_9 and G_{10} are identical, because $\mathcal{Z}$ never observes that d_K and SK were not derived from H_{d} and H_{SK}. Thus, $\Delta(\mathsf{G}_9, \mathsf{G}_{10}) \leq \Pr[\text{KQueried}]$. Furthermore, the only information $\mathcal{Z}$ sees about K is the commitment C_K, so the hiding property of ElGamal commitment should imply that it is hard for $\mathcal{Z}$ to compute K and trigger KQueried. Note that $\Pr[\text{KQueried}]$ is equal in G_9 and G_{10}, because the two games are identical up until the time KQueried occurs.

Given an environment $\mathcal{Z}$ that can trigger KQueried, we construct a reduction $\mathcal{R}$ that can solve DDH. The input to $\mathcal{R}$ is tuple $(A, B, C) \in \mathbb{G}^3$. $\mathcal{R}$ emulates game G_{10} to $\mathcal{Z}$ with the following modifications:

- In S, if $\mathbb{w}_{\mathsf{C}} \neq \perp$, then $\mathcal{R}$ parses $\mathbb{w}_{\mathsf{C}} \to (\mathsf{hpw}, s_{\mathsf{C}}, \cdot, \cdot, \cdot, \cdot)$. If $F = f^*_{\mathsf{PoNR}}(\mathsf{sid}_{\mathsf{S}}, \mathsf{hpw}, s_{\mathsf{C}} \oplus s_{\mathsf{S}})$, then $\mathcal{R}$ proceeds as before.
- Otherwise (i.e. $\mathbb{w}_{\mathsf{C}} = \perp$ or $F \neq f^*_{\mathsf{PoNR}}(\mathsf{sid}_{\mathsf{S}}, \mathsf{hpw}, s_{\mathsf{C}} \oplus s_{\mathsf{S}})$), instead of sampling $K \leftarrow_{\$} \mathbb{G}$, $C_K := \mathsf{ElG.Commit}(K; d_K)$, $\mathcal{R}$ samples $\delta, \mu, \nu \leftarrow_{\$} \mathbb{Z}_p$, $c_3 \leftarrow_{\$} \mathbb{G}$, and sets $C_K := (A \cdot g^{\delta}, g^{\mu} \cdot B^{\nu}, c_3)$. Also, $\mathcal{R}$ computes $K^* := c_3/(A^{\mu} \cdot C^{\nu}) \cdot (g^{\mu} \cdot B^{\nu})^{\delta}$ and adds $(\mathsf{sid}_{\mathsf{S}}, K^*)$ to set W.

Reduction $\mathcal{R}$ outputs 1 iff $\mathcal{Z}$ ever performs a query $\mathsf{H}_{\mathsf{d}}^{\mathsf{sid_S}}(K^*)$ or $\mathsf{H}_{\mathsf{SK}}^{\mathsf{sid_S}}(\mathsf{ssid}, K^*)$ (with any ssid) for any $(\mathsf{sid_S}, K^*) \in W$.

The views of $\mathcal{Z}$ in $\mathcal{R}$ and G_{10} are identical. In both games, each C_K is a commitment to a random K that is never otherwise used (referring only to the cases of responding to an honest client or to an unsuccessful dishonest client). In G_{10}, the game samples these K values and commits to them outright. In $\mathcal{R}$ the reduction uses (A, B, C) to create the commitments (again applying DDH randomized self-reduction).

If (A, B, C) is a Diffie-Hellman tuple, then the committed K is K^*, so the KQUERIED event is exactly the same as the condition for $\mathcal{R}$ to output 1. Thus, $\Pr[1 \leftarrow \mathcal{R} \,|\, (A, B, C) \in DH] = \Pr[\text{KQUERIED}]$. If (A, B, C) is not a Diffie-Hellman tuple, the committed K and K^* are independently random, and no information about K^* is exposed to $\mathcal{Z}$. At most q_{S} $(\mathsf{sid_S}, K^*)$ tuples are recorded, so $\Pr[1 \leftarrow \mathcal{R} \,|\, (A, B, C) \notin DH] \leq (q_{\mathsf{S}} q_{\mathsf{H}})/p$. The absolute difference between these two probabilities is $\leq \mathsf{Adv}_{\mathsf{DDH}}$.

$$\Delta(\mathsf{G}_9, \mathsf{G}_{10}) \leq \mathsf{Adv}_{\mathsf{DDH}} + \frac{q_{\mathsf{S}} q_{\mathsf{H}}}{p}$$

Game G_{11} (H_{SK} is programmed with SK when responding to successful dishonest clients): G_{11} is G_{10} with the following changes to S:

- If $\mathbb{w}_{\mathsf{C}} = \bot$, then proceed as before.
- Otherwise, parse $\mathbb{w}_{\mathsf{C}} \to (\mathsf{hpw}, s_{\mathsf{C}}, \cdot, \cdot, \cdot, \cdot)$. If $F \neq f^*_{\mathsf{PoNR}}(\mathsf{sid_S}, \mathsf{hpw}, s_{\mathsf{C}} \oplus s_{\mathsf{S}})$, then still proceed as before.
- Otherwise, do not compute C_K and V as normal. Instead, sample $\mathsf{SK} \leftarrow_{\$} \{0,1\}^{\kappa}$ and $K' \leftarrow_{\$} \mathbb{G}$. Then, compute $V \leftarrow \mathsf{EIG.Enc}_{\mathsf{pk_C}}(K')$ and $C_K \leftarrow \mathsf{EIG.Commit}(K'; \mathsf{H}_{\mathsf{d}}^{\mathsf{sid_S}}(K'))$. Finally, program $\mathsf{H}_{\mathsf{SK}}^{\mathsf{sid_S}}(\mathsf{ssid_S}, K') := \mathsf{SK}$.

The change from G_{10} to G_{11} is mostly conceptual and merely shifts the order in which values are sampled and computed. The only actual difference from the previous game is that SK is sampled independently of K' and then programmed into H_{SK}. Consequently, the views of $\mathcal{Z}$ in G_{10} and G_{11} are identical unless $\mathcal{Z}$ queries $\mathsf{H}_{\mathsf{SK}}^{\mathsf{sid_S}}(\mathsf{ssid_S}, K')$ before the game samples $K' \leftarrow_{\$} \mathbb{G}$ and programs it. Only one S session runs per $\mathsf{ssid_S}$, so only one H_{SK} point is programmed per $\mathsf{ssid_S}$, and the environment's probability of having guessed one of them is negligible.

$$\Delta(\mathsf{G}_{10}, \mathsf{G}_{11}) \leq \frac{q_{\mathsf{H}}}{p}$$

Game G_{12} (the simulated world): The change from G_{11} to G_{12} is purely conceptual. All interactions between the simulator and the honest parties now match the code of $\mathcal{F}_{\mathsf{CKEM}}$, so one can imagine the monolithic simulator now cleanly split into the two components $\mathcal{F}_{\mathsf{CKEM}}$ and $\mathsf{SIM}_{\mathsf{CKEM\text{-}PoNR}}$.

$$\Delta(\mathsf{G}_{11}, \mathsf{G}_{12}) = 0$$
$$\mathsf{Exp}_{\mathcal{Z}}[\mathsf{G}_{12}] = \mathbf{Ideal}_{\mathcal{Z}}[\mathcal{F}_{\mathsf{CKEM}}, \mathsf{SIM}_{\mathsf{CKEM\text{-}PoNR}}]$$

Summing up the distinguishing advantage bounds for each incremental game change yields an overall bound on $\mathcal{Z}$'s distinguishing advantage between the real and simulated worlds.

$$\Delta(\mathsf{G}_0, \mathsf{G}_{12}) \leq 3 \cdot \mathsf{Adv}_{\mathsf{DDH}} + \mathsf{Adv}_{\mathsf{DL}} + \frac{{q_{\mathsf{C}}}^2 + {q_{\mathsf{S}}}^2 + q_{\mathsf{S}} q_{\mathsf{H}} + q_{\mathsf{H}}}{p}$$

$p = |\mathbb{G}|$ is an exponential function of the security parameter. If $\mathcal{Z}$ is efficient, then q_{C}, q_{S}, and q_{H} are polynomial functions of the security parameter. Furthermore, $\mathsf{Adv}_{\mathsf{DDH}}$ and $\mathsf{Adv}_{\mathsf{DL}}$ are negligible functions of the security parameter under the DDH assumption. Thus, the distinguishing advantage of any efficient $\mathcal{Z}$ between the real and simulated worlds is negligible.

Protocol $\mathsf{CKEM}_{\mathsf{PoNR}}$ realizes $\mathcal{F}_{\mathsf{CKEM}}[\mathsf{SSEM}[f^*_{\mathsf{PoNR}}, \mathsf{EIG}], \mathsf{SSEM}_{\mathsf{Adv}}[f^*_{\mathsf{PoNR}}]]$.