Encryption with Untrusted Keys: Security against Chosen Objects Attack

Shashank Agrawal[1], Shweta Agrawal[2], and Manoj Prabhakaran[3]

[1] Visa Research. Email: shashank.agraval@gmail.com.
[2] Indian Institute of Technology Madras. Email: shweta.a@cse.iitm.ac.in.
[3] Indian Institute of Technology Bombay. Email: mp@cse.iitb.ac.in.

Abstract. In Public-Key Encryption, traditionally no security is expected if honest parties use keys provided by an adversary. In this work, we re-examine this premise. While using untrusted keys may seem nonsensical at first glance, we argue the use of providing certain security guarantees even in such situations. We propose *Chosen Object Attack* (COA) security as a broad generalization of various notions of security that have been considered in the literature, including CCA security, key anonymity and robustness, along with concerns arising from untrusted keys. The main premise of this definition is that any of the objects in a cryptographic scheme could be adversarialy generated, and that should not compromise the security of honest parties in a way an idealized scheme would not have.
Our contributions are threefold.

- Firstly, we develop a comprehensive security definition for PKE in the real/ideal paradigm. Our definition subsumes CCA2 security, Anonymity and Robustness as special cases, and also addresses security concerns in complex application scenarios where the keys may be malicious (without having to explicitly model the underlying attack scenarios). To avoid impossibility results associated with simulation-based security, we use the notion of *indistinguishability-preserving* security (IND-PRE) from the "Cryptographic Agents" framework (Agrawal et al., EUROCRYPT 2015). Towards this, we extend this framework to accommodate adversarially created objects. Our definition can alternately be interpreted as the union of *all possible game-based security definitions.*
We remark that the agents framework as extended in this work is applicable to primitives other than Public-Key Encryption, and would be of broader significance.
- Secondly, and somewhat surprisingly, we show that in the case of PKE, the above comprehensive definition is implied by a simpler definition (which we call COA security) that combines a traditional game-based definition with a set of consistency requirements. The proof of this implication relies on an extensive analysis of all possible executions involving arbitrarily many keys and ciphertexts, generated, transferred between parties and used in an arbitrary and adaptive manner.
- Thirdly, we consider constructions. Interestingly, using the above security definition, we show that the Cramer-Shoup cryptosystem (with minor modifications) already meets our definition. Further, we present transformations from any Anonymous CCA2-secure PKE scheme to a COA-secure PKE. Under mild correctness conditions on the Anonymous CCA2-secure PKE scheme, our transformation can be instantiated quite efficiently and is arguably a viable enhancement for PKE schemes used in practice.

1 Introduction

Today, we live in an intricate web of digital objects and activities, amidst a plethora of security concerns. The rise of *mobile apps*, *cloud computing*, the *Internet of Things* and *blockchains* have all changed the landscape of security threats. In this context, we propose revisiting one of the simplest and most fundamental cryptographic primitives, namely, encryption: Can the security guarantees for Public-Key Encryption (PKE) be strengthened so that it can be useful in new usage scenarios? Technically, we investigate the following fundamental question:

What security guarantees are needed when honest parties may run the algorithms in an encryption scheme on adversarially generated inputs, including messages, ciphertexts and public/secret keys?

While there has been a significant amount of literature on understanding and strengthening security definitions for encryption, all these works invariably require that *the secret-keys used by an honest party are generated honestly.* But, several modern usage scenarios violate this assumption. For instance, an app or a device may download a decryption key from its developer or third party services, or a party may obtain a decryption key that is published as part of a smart contract on a blockchain.

Our contributions in this work fall broadly into two parts:

• *Defining Security.* Using untrusted keys in complex scenarios can have subtle consequences beyond what one may intuitively expect (see later for some example scenarios). Hence a security definition that extends the use of encryption to scenarios involving untrusted keys must use a comprehensive model with arbitrary use of an idealized encryption primitive. Indeed, in this work we put forth such a model and definition, namely, an *Extended Cryptographic Agents* model, and a public-key encryption *schema* in this model.

• *Meeting the Definition.* Instead of presenting one construction which achieves our comprehensive security definition, we present a general result that any construction with a collection of simpler properties meets the above definition. (This is the most technically challenging part of this work.) We collect these properties under the name of "security against Chosen Objects Attack," or *COA-security*, for short. We go on to show that COA-security can be achieved from PKE schemes with CCA-security and *anonymity* [4], with low overheads.

Framework Approach vs. Focused Approach. Security definitions for primitives evolve over time. In the case of PKE, for instance, the need to consider maliciously created ciphertexts – or chosen ciphertext attacks – was discussed from the early days of public-key encryption (PKE) [29], and over the years led to development of IND-CCA2 security (referred to simply as CCA security, in the sequel) as the standard security goal for encryption schemes [8,28,15,7]. Some more important concerns regarding non-ideal behavior of encryptions have received attention, like *anonymity* (whether a ciphertext reveals the identity of the receiver's public key) [4,21] and more recently *robustness* (what should happen if a user decrypts anonymous ciphertexts meant for someone else) [1,27,17] (see Appendix A for more details). These issues, specific to PKE, were discovered as more and more scenarios involving PKE were studied over the years. This approach to security definitions could be called "focused," as it is guided by a careful study of a specific primitive and its usage.

An alternate approach is what one might call the "framework approach." A wonderful example of the framework approach is the Universal Composition framework [9], which followed several years of advances in our understanding of composition that emerged from the focused approach (e.g., notions like non-malleability [15] and concurrent zero-knowledge proofs [16]). While the focused approach may often highlight major issues, security definitions closely based on specific threats leave open the possibility of overlooking many other subtle vulnerabilities. That is, a security definition designed to address specific concerns is prone to "overfitting." Instead, the security definitions in the framework approach are agnostic about individual attacks, and are more likely to "generalize" to other unforeseen attacks. This is all the more important in complex scenarios where security requirements tend to be less intuitive (as is the case when all components including the keys can be adversarial).

The current work follows the framework approach: Our security definitions and results on PKE fall out of a framework that goes well beyond PKE.

Do We Need Yet Another Definition for PKE? PKE is a remarkably well-studied primitive, with strong security definitions. In this context, one may ask if we really *need* stronger security definitions. There are a couple of different reasons why we do.

◇ *Stronger Security Enables New Applications.* The original semantic security (IND-CPA) definition [20] is adequate for establishing secure communication channels on top of authenticated channels[4] but for an application like encrypted e-mail, the stronger IND-CCA security is required. Anonymity and robustness were similarly identified as enabling other natural applications of (ideal) encryption schemes. Similarly, our definitions are motivated by the possibility of a plethora of complex scenarios which need to retain some level of security even when various keys in the system cannot be fully trusted.

[4] This is so because the PKE keys can all be ephemeral – i.e., freshly generated in each session – with long-term keys used only for authentication.

⋄ *Bridging Popular Notions and Actual Cryptographic Guarantees.* Real-world security vulnerabilities lurk in the *gaps between what popular ideal notions of cryptographic primitives promise and what the actual security definitions guarantee*, and hence minimizing this gap is important. Primitives like PKE find their way into widely used applications and standards that are often designed heuristically (and analyzed formally later on, if at all), based on an intuitive understanding of the security offered by an idealized version of the primitives. From an academic point of view, one may simply consider such designs indefensible. On the other hand, by providing primitives which are closer to their idealized versions, we may minimize the real-world security risk introduced by such applications.[5]

Scenarios that Need Security Against Malicious Keys. We present a few examples of unexpected consequences of operating on malicious keys, that are not addressed by definitions like anonymity (Anon-CCA) and robustness. These scenarios do not occur in the most traditional uses of PKE (establishing secure channels, encrypting e-mails, etc.), but may need to be considered if PKE has to deployed in more complex scenarios.

– A third party could *censor* the communication between honest parties by providing the sender and receiver with maliciously crafted public/secret keys, such that ciphertexts carrying certain messages will not be decrypted. While the honest parties cannot expect the ciphertexts to be hiding against the adversary (which may not be a concern, e.g., if the adversary does not have access to the ciphertext), they may reasonably expect that the messages will be delivered unaltered. However, standard encryption gives no guarantees of correctness when malicious secret-keys are used for decryption.

– A third party could present two different secret keys to two parties such that when an honestly generated ciphertext is posted, they both proceed to decrypt it differently. The adversary may also be able to create tailor-made ciphertexts which will decrypt to specific different messages for the two parties. A variant of this attack is when the adversary observes a valid public-key posted online, and generates a fake secret-key for it, which "decrypts" ciphertexts generated using that public key into plausible messages.
We remark that "complete robustness" was proposed as a means to consider malicious public-keys [17] (strengthening the original notion of (strong) robustness [1,27] which only addressed the case of honestly generated keys). However, this definition only prevents the adversary from "explaining" the generation of a single ciphertext as coming from two *different* public-keys, and does not preclude the above attacks involving a single public-key and different secret-keys.

– An adversary may generate multiple seemingly unrelated public-keys that all produce ciphertexts that correctly decrypt under the secret-key of a given public-key.[6] This can lead to unexpected attacks that are absent with an idealized encryption scheme (see Appendix B for an illustration).

– As a dual of the above attack, an adversary may generate a single public-key which produces ciphertexts which decrypt under various honest parties' secret-keys (whose public-keys the adversary had access to). Further, which ciphertexts produced using this malicious public-key are decrypted (correctly or incorrectly) by a given secret-key could depend on the message (and randomness) used during encryption.

Choice of Security Framework Modern cryptography offers a powerful mechanism for comprehensive security definitions, namely, the "real/ideal paradigm" of simulation-based security [19]. This approach has formed the foundation for general frameworks like Universally Composable security [9] and Constructive Cryptography [26]. Unfortunately, simulation based security definition (in the standard model, without say, random oracles) turns out to be unachievable in our case. Indeed, when secret-keys can be transferred adaptively, simulation-based security is impossible to achieve, even for symmetric-key encryption (see Appendix H).[7]

[5] To draw an analogy, traditionally cryptography assumes a model with no side-channels. One may consider real-world implementations with significant side-channels to be outside the scope of cryptographic guarantees, or alternately, extend the theory to cover such leakage.

[6] Waters et al. [32] considered allowing such an attack as a *feature* (with all the public-keys generated along with the secret-key). For us, this is a vulnerability that we seek to remove.

[7] Simulation-based security against key exposure is possible for one-time encryption [10]. While this suffices in the context of secure computation protocols, this is unsatisfactory for PKE wherein the same secret-key should allow decrypting an *a priori* unbounded number of ciphertexts (possibly sent by different parties).

On the other hand, the traditional game-based (IND) security framework does not offer a real/ideal security guarantee for an arbitrary environment,[8] but requires designing the security experiment to explicitly address specific security concerns (chosen plaintext attack, chosen ciphertext attack, anonymity, robustness etc.). This leads us to a central technical challenge:

> *We desire a security definition for encryption that* (1) *models adaptively transferable – and possibly adversarially generated – objects (messages, ciphertexts, and keys), and* (2) *uses the real/ideal paradigm, but* (3) *is not subject to the impossibility results for simulation-based security definition. We may also require (informally) that* (4) *it should not be overly tedious to prove that a construction meets the security definition.*

A possible approach to resolving the tension between (2) and (3) above appears in the recently suggested "Cryptographic Agents" model [2], where *indistinguishability preservation* (IND-PRE) was put forth as a security guarantee in the real/ideal framework. IND-PRE security only gives an indistinguishability security guarantee in environments which already provide the same indistinguishability guarantee in the ideal world. Indeed, the version of IND-PRE security we use, Δ-s-IND-PRE security, could be interpreted as *the universal IND security definition, which accommodates all possible experiments that can be used in IND security definitions.* As such, this would be an ideal solution to the above problem.

Unfortunately, we cannot directly use the model proposed in [2], as it does not address the condition (1) above, since it cannot model adversarially created cryptographic objects (and cannot be used to model CCA security, as explained in [2], let alone adversarially generated keys). Hence, we propose to extend the cryptographic agents model with a facility for the adversary to generate cryptographic objects and transfer them to the honest users. This yields the main security definition we use in this work, which we shall continue to refer to as Δ-s-IND-PRE security (even though the model has been extended).

However, it is fair to say that this definition fails requirement (4) in the above desiderata: Proving that a construction meets this definition is extremely tedious, as it requires analyzing complex executions in an arbitrary environment with the corrupt and honest parties running key-generation, encryption and decryption algorithms on inputs possibly created by other parties. To address this, we develop a simpler definition called *Chosen Object Attack* (COA) security for PKE – which combines an IND security definition (called Anon-CCA security, which is equivalent to IND-CCA and IK-CCA [4] combined) with a list of carefully chosen correctness guarantees – and show that COA security implies Δ-s-IND-PRE security in the extended Cryptographic Agents framework. On the face of it, the correctness guarantees in COA security may not seem to address all possible combinations of maliciously created objects.[9] Not surprisingly, proving that COA security implies the Δ-s-IND-PRE security definition turns out to be quite tedious, but in return we obtain a definition *with the simplicity of IND definitions and the generality of the real/ideal definitions.*

Cryptographic Agents. We briefly review the Cryptographic Agents framework here (with more technical details appearing in Section 5). Cryptographic Agents were originally proposed as a framework to model various cryptographic objects ranging from modern primitives such as fully-homomorphic encryption, functional encryption, obfuscation to classic primitives such as public key encryption to generic group and random oracle models [2].

The framework is conceptually simple: there is an *ideal model* in which a trusted party hands out handles to users for manipulating data stored with it. The manner in which data can be manipulated in the ideal model is specified by a "schema" (which is akin to a functionality in the UC security model). In the *real*

[8] IND-CCA and even IND-CPA security do imply UC secure communication primitives, provided that the honest parties only use the keys they locally generate, and never reveal their keys to the adversary. Our focus is on settings where such restrictions do not apply.

[9] For instance, COA security does not prevent "proxy reencryption": the adversary may be able to generate two key pairs (SK_1, PK_1) and (SK_2, PK_2) such that a ciphertext corresponding to PK_1 can be modified (without knowing SK_1) into a ciphertext corresponding to PK_2. (Note that this particular "attack" is arguably not problematic, since an honest party shall not carry out such a modification operation, and the adversary could anyway carry out such a modification by decrypting and reencrypting, given that it had the opportunity to create the keys.)

model cryptographic objects are used in place of the ideal handles. The real world is required to hide any predicate that is hidden by the ideal world, as formalized by the "IND-PRE" (indistinguishability preservation) guarantee.

The Cryptographic Agents framework is an attractive choice for us because of a confluence of multiple features: (1) security is defined in an ideal-real paradigm, involving the interaction of a Test and a User, (2) it naturally models all objects – ciphertexts, public keys and secret-keys, in the case of PKE – as transferable, and (3) an indistinguishability style security guarantee is employed so as to bypass the impossibilities that a simulation style definition would suffer from. Specifically, for a "test-family" Γ, we define Γ-IND-PRE security to hold if for every Test $\in \Gamma$ that keeps its input hidden from a (possibly corrupt) User in the ideal world, Test keeps its input hidden in the real world too. The Γ-s-IND-PRE is a variant of the definition that restricts the security guarantee to Tests in Γ that are hiding against even computationally unbounded Users in the ideal world [3].

In the Cryptographic Agents framework, in the ideal world, the handles give blackbox access to idealized *agents*, which may carry secrets, interact with each other, possibly evolve, and may reveal prescribed information when invoked with inputs or made to interact with other agents. The schema specifies how these agents behave. We use a fairly straightforward formulation of the PKE schema $\boldsymbol{\Sigma}_{\mathsf{pke}}$, but making explicit the guarantees we *do not* seek (e.g., we allow an adversary with one secret-key for a public-key to generate more). We remark that our security guarantees are applicable only when honest parties use the encryption scheme through the standard interface which restricts them to key generation, encryption and decryption. We shall not address applications of encryption that go beyond this interface. In particular, an ad hoc usage of PKE for *commitment*, with opening carried out by revealing the randomness used for encryption, is not covered by Δ-s-IND-PRE security or COA security.

Extension of Cryptographic Agents. In the original model of [2], all the agents were created by Test and transferred to the user. In the real world, this meant that all cryptographic objects would be honestly generated. In particular, it was already pointed out in [2] that CCA secure encryption could not be modeled in their framework as it involved adversarially generated ciphertexts. An important contribution of this work is to address this limitation of the original framework, and formulate an extended model in which Test and User (who can be corrupt) can both create objects and transfer them to each other. See Section 2.2 for further details.

$\boldsymbol{\Delta}$-s-IND-PRE Security. The strongest s-IND-PRE definition obtained in the Cryptographic Agents framework by allowing Test to be any arbitrary probabilistic polynomial time (PPT) program results in a definition that is impossible to realize (even for symmetric key encryption and even in the original framework of [2] – see Appendix H). However, a more restricted test-family called Δ suffices to imply all possible IND-style definitions.[10]

Informally, a Test $\in \Delta$ reveals everything about the agents it sends to User except for a test-bit b. When transferring an agent to User, Test chooses two handles h_0, h_1 and communicates these to the user but *transfers* only h_b where b is the test-bit. Thus, User knows that Test has transferred one of two known agents to her, but does not know which. User may proceed to perform any idealized operation with this newly transferred handle.

Δ-s-IND-PRE security *subsumes essentially all meaningful IND security definitions*: for any such IND security game, there is Test $\in \Delta$ which carries out this game, such that it statistically hides the test-bit

[10] Here an IND security definition refers to an experiment in which an adversary sends instructions to an experiment (key-generation, encryption, decryption, and transfers) to be applied on inputs (messages, ciphertexts, public or private keys) that she specifies; the inputs can be explicit or handles referring to the outputs of prior instructions. The adversary may also receive arbitrary side information from the environment, as specifed by the experiment (e.g., first bit of a decrypted message). Some of the instructions from the adversary can be in terms of a secret bit (test-bit) b that is uniformly sampled at the beginning of the experiment (e.g., send an encryption of m_b from (m_0, m_1)). The security requires that the adversary has negligible advantage in guessing this bit as long as she could not have trivially inferred it without "breaking" the encryption: i.e., if an ideal encryption scheme were to be used, b would be statistically hidden.

when an ideal encryption scheme is used (e.g., in the case of IND-CCA security this formulation corresponds to a game that never decrypts a ciphertext that is identical to the ciphertext that was earlier given as the challenge, called IND-CCA-SE in [6]), and Δ-s-IND-PRE security applied to this Test translates to the security guarantee in the IND security game. In intuitive terms, Δ-s-IND-PRE formalizes the following guarantee: *as long as* Test does not reveal a secret (represented by the test-bit[11]) in the ideal world, the real world will also keep it hidden.

In particular, Δ-s-IND-PRE security directly addresses the chosen object attacks of the kind addressed earlier, as they can all be captured using specific IND security games. On the other hand, we show that Δ-s-IND-PRE security is implied by COA security. Thus the seemingly simple consistency requirements posited in the COA security definition do provably rule out violating any secrecy guarantee that exists when an idealized encryption scheme is used.

Limitations of Δ-s-IND-PRE *security.* Even though Δ-s-IND-PRE security is based on an ideal world model, and subsumes *all possible* IND definitions (see Footnote 10), we advise caution against interpreting Δ-s-IND-PRE security on par with a simulation-based security definition (which, indeed, is unrealizable). Firstly, Δ-s-IND-PRE does not require preserving non-negligible probabilities: e.g., an event with probability at most $\frac{1}{2}$ in the ideal world could have probability $\frac{3}{4}$ in the real world. A related issue is that while an ideal encryption scheme could be used as a non-malleable commitment scheme, Δ-s-IND-PRE security makes no such assurances. This is because, in the ideal world, if a commitment is to be opened such that indistinguishability ceases, then IND-PRE security makes no more guarantees. We leave it as an intriguing question whether Δ-s-IND-PRE secure encryption could be leveraged in an indirect way to obtain a non-malleable commitment scheme (perhaps analogous to how indistinguishability obfuscation (iO) could be leveraged to obtain security guarantees involving non-identical programs for which iO does not provide direct guarantees).

2 Technical Overview

2.1 COA Secure Encryption: Definition and Construction

The definition of COA security is deceptively simple. It consists of two parts: Anonymous CCA (or Anon-CCA) security and "existential consistency." The latter is a natural correctness guarantee that requires that even an adversarially generated ciphertext should have at most one message and one public-key associated with it, and even maliciously generated secret-keys will decrypt it in a manner consistent with its underlying public-key and message.

More formally, there is an efficient algorithm used to accept or reject externally generated objects (keys, ciphertexts). For any object that is accepted as a secret key, there should be a deterministic procedure, which we denote by pkGen, to convert it to a public key. Additionally, for any object that is accepted as a ciphertext, there must be an information theoretic binding of the ciphertext to a (hidden) public key and message, captured by the existence of (computationally intractable) maps $\mathsf{pkId} : \mathcal{CT} \to \mathcal{PK} \cup \{\bot\}$ and $\mathsf{msgId} : \mathcal{CT} \to \mathcal{M} \cup \{\bot\}$. The consistency requirement insists that for any *matching* secret key and ciphertext, namely, when $\mathsf{pkGen}(SK) = \mathsf{pkId}(CT)$, the decryption procedure must always reveal $\mathsf{msgId}(CT)$ (which may be $\bot$), and if the key and ciphertext do not match, it must always output $\bot$.

We present two general constructions for a COA secure PKE scheme, by modifying an arbitrary Anon-CCA encryption scheme. The first construction is fairly light-weight, and considering that it can be used in the hybrid encryption (KEM/DEM) mode, quite efficient. It relies on a slightly non-standard correctness requirement (which we call universal key reliability), which states that for *all* secret-keys that can be generated by the key generation algorithm, honestly encrypting any message with its corresponding public-key and then decrypting the resulting ciphertext should return the original message, with high probability (over the randomness used for encryption).

A helpful first step in preventing invalid secret-keys is to redefine it to be the randomness used to generate the original secret-key. Further towards enforcing existential consistency, we augment the public-key to include

[11] We note that this need not be a single bit, and may generalize to allow for polynomially many possibilities.

a statistical commitment to the secret-key, and the ciphertext is augmented to include a commitment to the public-key. That is, the ciphertext has the form (α, β), where α is a commitment to the public-key and β is a ciphertext in the original scheme. In order to preserve Anon-CCA security, the message that is encrypted in β itself is augmented with α (and its decommitment information). The details of this construction are given in Section 4.1.

The proof of Anon-CCA security is not immediate. We carry it out in two parts. First, we show that the construction is CCA secure. Then, to prove Anon-CCA security, among other things we need to address what happens when the adversary takes the challenge ciphertext (α, β) and queries the *wrong* decryption oracle with (α', β). Here we rely on the fact that our augmented public-key has high min-entropy (from the commitment that is included in it) to rule out the possibility that β accidentally decrypts (under the wrong secret-key) to yield α' which is a statistically binding commitment to the (wrong) public-key.

Our second construction is perhaps of more theoretical interest as it proves the following theorem.

Theorem 1. *A COA secure PKE scheme exists if an* Anon-CCA *secure PKE scheme and injective one-way functions exist.*

Note that here we do not require the universal key reliability property of the given PKE scheme. However, this significantly complicates the construction (and the proof). The first challenge now is to weed out potentially bad secret-keys which may decrypt *some* message incorrectly with a significant probability. Since it will be difficult to detect such a worst case behavior, we randomize the message (via secret-sharing) before encrypting. Now, a probabilistic check can ensure that a secret-key causes decryption error with at most an inverse polynomial error probability. However, to achieve COA security, we need to drive this probability down to a negligible quantity. This can be achieved by including multiple encryptions, and relying on error-correction during decryption.

However the use of secret-sharing and error-correction create several complications with CCA security, let alone Anon-CCA security. While CCA security can be restored by carefully using a signature scheme, key anonymity requires further work. In particular, we need to analyze what happens when, in the given PKE scheme pke, a wrong key is used to decrypt a ciphertext. One might expect that such a mismatched decryption will result in "garbage" and will be of little use to the adversary in creating a decryption query in the Anon-CCA game. Unfortunately, this intuition is wrong: while the adversary cannot control the outcome of decrypting an honestly generated ciphertext using an honestly and independently generated secret-key in the given PKE scheme, it is possible that this outcome is *predictable* (and not $\perp$). As this decryption yields only a share of a message, being able to predict it allows the adversary to control the reconstructed message. To counter this, we require that each of the shares carry a tag that was randomly chosen and included in the public-key, so that again, the adversary will need to control the outcome of mismatched decryption. The details of this construction are given in Section 4.2.

2.2 Extending the Cryptographic Agents Model

As discussed above, cryptographic agents [2] provides a framework that naturally models all cryptographic objects (keys as well as ciphertexts, in the case of encryption) as transferable. However, the framework as defined in [2] does not capture attacks involving maliciously created objects. In the case of encryption, even CCA security could not be modeled in this framework.

We make several technical extensions to allow modeling COA security. We highlight the important ones below.

• Firstly, we use an execution model that treats Test and User symmetrically, allowing both parties to transfer agents (or objects in the real world) to each other. This automatically allows for the possibility that the objects in the real world – including secret-keys as well as public-keys and ciphertexts – could be created maliciously (as the User can be corrupt).

• Secondly, we introduce a mechanism that allows the two parties to locally act on the agents in their possession, and only selectively transfer agents to each other (in contrast, in [2], all agents created by Test would get automatically transferred to User, while the latter could not transfer any agent to Test). This models, in particular, various operations that can be executed by honest parties on objects received from the adversary.

• In [2] encryption-like primitives were modeled so that only a single key-agent existed in the system. In our formulation, we model the agents in an encryption scheme as evolving from a secret-key agent, which is initialized using a randomized initialization step. Such an initialization, which was not part of the original framework, allows us to model multiple keys in the system in a sound manner (by including random tags generated during initialization, which are not controlled by Test or User).

• In our new model, we introduce a mechanism to "vet" an object before accepting it. This opens up new avenues in constructing schemes that securely implement various schemas.

• Following [3], we slightly relax the security definitions in [2] so that indistinguishability holds in the real world only if in the ideal world indistinguishability holds against computationally unbounded adversaries. This relaxation turns out to be crucial in exploiting existential consistency of COA security to argue that COA security implies a Δ-s-IND-PRE secure implementation of the PKE schema.

2.3 Proving that COA Security implies Δ-s-IND-PRE Secure PKE

Implementing the schema Σ_{pke} is a challenging task because it is highly idealized and implies numerous security guarantees that may not be immediately apparent. (For instance, the ideal world provides "ciphertext resistance" in that an adversary who gets oracle access to encryption and decryption cannot create a valid ciphertext that it did not already receive from the encryption oracle.) These guarantees are not explicit in the definition of COA security. Nevertheless, we show the following:

Theorem 2. *A Δ-s-IND-PRE secure implementation of Σ_{pke} exists if a COA secure PKE scheme exists.*

While the construction itself is direct, the proof is much less so. We use a careful sequence of hybrids to argue indistinguishability preservation. The hybrids involve the use of an "extended schema" (which is partly ideal and partly real) and simulators which help one switch from the real world to an ideal world. We use both PPT simulators (which rely on Anon-CCA security) and computationally unbounded simulators (which rely on existential consistency). They heavily rely on the fact that Test $\in \Delta$, and hence the only uncertainty regarding agents transferred by Test is the choice between one of two known agents, determined by the test-bit b given as input to Test. The essential ingredients of these simulators are summarized below. (An outline of the steps are also shown in Section 7.2.)

• One can move from the real execution to an execution in which objects originating from Test are replaced by ideal agents, while the objects originating from the adversary are left as such (in the form of non-ideal agents, which carry the real objects within). The extended schema is used to process both kinds of agents together. This hybrid uses a simulator $\mathcal{S}_b^{\dagger}$ which knows exactly the test bit b. Since Test $\in \Delta$ reports all its interactions with the ideal schema, $\mathcal{S}_b^{\dagger}$ can exactly simulate all the objects created by Test. Here one needs to be careful in sorting objects as originating from Test or adversary, because if the adversary uses a public-key sent by the Test to create a ciphertext, this should be treated as an object originating from Test and replaced by an ideal agent. (The two hybrids here, corresponding to $b = 0$ and $b = 1$ are called H_1 and H_6 in Section 7.2. The above sketched argument establishes $\mathsf{H}_0 \approx \mathsf{H}_1$ and $\mathsf{H}_6 \approx \mathsf{H}_7$, where H_0 and H_7 correspond to real executions.)

• The next step is to show that there is a simulator $\mathcal{S}^{\ddagger}$ which does not need to know the bit b to carry out the above simulation. This is the most delicate part of the proof. The high-level idea is to argue that the executions for $b = 0$ and $b = 1$ should proceed identically from the point of view of the adversary (as Test hides the bit b in the ideal world), and hence a joint simulation should be possible. Unlike $\mathcal{S}_b^{\dagger}$ (for either value of b) $\mathcal{S}^{\ddagger}$ assigns objects to agents only when they are transferred by Test. We expect that the assignment made by $\mathcal{S}^{\ddagger}$ can be extended to an assignment by $\mathcal{S}_b^{\dagger}$ for either value of b. When a single object cannot be assigned to both the agents, $\mathcal{S}^{\ddagger}$ will abort, and we shall argue that whenever this happens, it corresponds to revealing b in the ideal execution.

The actual argument is more complicated. An example of a complication that arises is that Test might transfer a ciphertext agent, such that it has different messages in the two executions corresponding to $b = 0$

and $b = 1$. Then there is no consistent assignment of that agent to an object that works for both $b = 0$ and $b = 1$. Nevertheless this may still keep b hidden, as long as the corresponding secret-keys are not transferred. So $\mathcal{S}^\ddagger$ can assign a random ciphertext to this agent; provided that the key will be "locked away" and never transferred, $\mathsf{Anon\text{-}CCA}$ ensures that the simulation is good. To analyze this, we let $\mathcal{S}^\ddagger$ maintain a list of secret-keys that get locked in this way, and will abort the simulation if one of those keys is transferred later. Once $\mathcal{S}^\ddagger$ is carefully specified, it can be verified that if it aborts with non-negligible probability then the bit b was not hidden in the ideal world to begin with; otherwise, due to $\mathsf{Anon\text{-}CCA}$ security, the simulation is good. Here, b not being hidden does not yield a contradiction yet. (The hybrids above are called H_2 and H_5. The argument sketched above shows that *if* $\mathsf{H}_2 \approx \mathsf{H}_5$, *then* $\mathsf{H}_1 \approx \mathsf{H}_2$ and $\mathsf{H}_5 \approx \mathsf{H}_6$).

• The next simulator $\mathcal{S}^*$ is computationally unbounded, and helps us move from the ideal world with the extended schema to the ideal world involving only the schema Σ_{pke}. The key to this step is existential consistency: $\mathcal{S}^*$ will use unbounded computational power to map objects sent by the adversary to ideal agents. (The hybrids in this step are called H_3 and H_4, and the above argument establishes $\mathsf{H}_2 \approx \mathsf{H}_3$ and $\mathsf{H}_4 \approx \mathsf{H}_5$).

• To prove Δ-$\mathsf{IND\text{-}PRE}$ security we need only consider $\mathsf{Test} \in \Delta$ such that the bit b remains hidden against a *computationally unbounded* adversary. For such a Test, the above two hybrids are indistinguishable from each other ($\mathsf{H}_3 \approx \mathsf{H}_4$). Hence, the previous two hybrids are indistinguishable from each other ($\mathsf{H}_2 \approx \mathsf{H}_5$). This, combined with the above steps, establishes that the bit b is hidden in the real execution ($\mathsf{H}_0 \approx \mathsf{H}_7$), provided it is statistically hidden in the ideal execution.

3 Chosen Object Attack (COA) Secure Encryption

Let skGen, pkGen, $\mathsf{encrypt}$, and $\mathsf{decrypt}$ denote the algorithms for a public-key encryption scheme, and let $\mathcal{SK}$, $\mathcal{PK}$, $\mathcal{CT}$ and $\mathcal{M}$ denote the space of secret keys, public keys, ciphertexts and messages, respectively.[12] We require these spaces to be mutually disjoint, with efficient membership algorithms (as could be readily enforced, say by requiring a header). Here we use the convention that skGen is the algorithm for secret-key generation and pkGen converts secret-keys to public-keys. $\mathsf{pkGen} : \mathcal{SK} \to \mathcal{PK}$ and $\mathsf{decrypt} : \mathcal{SK} \times \mathcal{CT} \to \mathcal{M} \cup \{\perp\}$ are deterministic functions. (See Appendix D.1.)

Anonymous CCA Security. We use a security definition for key and message privacy against chosen-ciphertext attacks, called $\mathsf{Anon\text{-}CCA}$. It is equivalent to AI-CCA presented in [1] as a combination of IND-CCA and IK-CCA security, which was introduced in [4]. The experiment for $\mathsf{Anon\text{-}CCA}$ is similar to that of IND-CCA, except that the adversary is given two independently generated public-keys, and access to the decryption oracles corresponding to both of them. A random bit is used to select one of the keys as well as one in a pair of messages submitted by the adversary, to produce a challenge ciphertext (which cannot be queried to either decryption oracle). The formal definition is given in Appendix D.

Existential Consistency. Informally, the consistency requirement is that when the objects given by an adversary are operated on by the algorithms in the encryption scheme, they should behave as objects that were generated honestly according to an ideally correct scheme. Since we would like to consider non-uniform adversaries, this means that all objects should behave consistently with underlying ideal objects. We capture this in the following definition.

Definition 1. *An encryption scheme* $\mathsf{pke} = (\mathsf{skGen}, \mathsf{pkGen}, \mathsf{encrypt}, \mathsf{decrypt})$ *with message space* $\mathcal{M}$ *is said to be* existentially consistent *if the following conditions hold:*

1. *There is a PPT algorithm* accept *which takes any string as input and outputs one of the tokens* $\{\text{SK}, \text{PK}, \text{CT}, \perp\}$*, such that the following probabilities are negligible:*

$$\begin{aligned} &\Pr[\mathsf{accept}(\mathsf{skGen}()) \neq \text{SK}] && \\ &\Pr[\mathsf{accept}(\mathit{obj}) = \text{SK} \ \wedge \ \mathsf{accept}(\mathsf{pkGen}(\mathit{obj})) \neq \text{PK}] && \forall \mathit{obj} \\ &\Pr[\mathsf{accept}(\mathit{obj}) = \text{PK} \ \wedge \ \mathsf{accept}(\mathsf{encrypt}(\mathit{obj}, m)) \neq \text{CT}] && \forall \mathit{obj}, m \in \mathcal{M} \end{aligned}$$

[12] Not all the elements in a space may be *valid*, i.e., there could be an element in $\mathcal{SK}$, for instance, which is never output by skGen irrespective of the randomness used.

2. *There exist (computationally inefficient) deterministic functions* $\mathsf{pkId} : \mathcal{CT} \to \mathcal{PK} \cup \{\perp_{\text{PK}}\}$ *and* $\mathsf{msgId} : \mathcal{CT} \to \mathcal{M} \cup \{\perp\}$ *such that the following holds. Let* $D : \mathcal{SK} \times \mathcal{CT} \to \mathcal{M} \cup \{\perp\}$ *be defined as*

$$D(SK, CT) = \begin{cases} \mathsf{msgId}(CT) & \textit{if } \mathsf{pkGen}(SK) = \mathsf{pkId}(CT) \neq \perp_{\text{PK}} \\ \perp & \textit{otherwise.} \end{cases}$$

Then $\forall\ SK \in \mathcal{SK}, PK \in \mathcal{PK}, CT \in \mathcal{CT}, m \in \mathcal{M}$, *the following probabilities are negligible.*

$$\Pr[\mathsf{accept}(SK) = \text{SK} \ \wedge\ \mathsf{accept}(CT) = \text{CT} \ \wedge\ \mathsf{decrypt}(SK, CT) \neq D(SK, CT)]$$
$$\Pr[\mathsf{accept}(PK) = \text{PK} \ \wedge\ \mathsf{pkId}(\mathsf{encrypt}(PK, m)) \neq PK\}]$$
$$\Pr[\mathsf{accept}(SK) = \text{SK} \ \wedge\ \mathsf{msgId}(\mathsf{encrypt}(\mathsf{pkGen}(SK), m)) \neq m].$$

The intention of providing the algorithm accept is to run it on an object when it is received, in order to sort it as a secret-key, public-key or ciphertext (or none of them). This will be made explicit in the interface provided in the cryptographic agents model. In typical schemes, accept could simply correspond to a format check (possibly reading headers and checking length). However, as we shall see in Section 4.2, a non-trivial probabilistic check can sometimes be useful to vet an object before accepting it.

Definition 2. *A public-key encryption scheme is said to be* COA-secure *if it is* Anon-CCA *secure and existentially consistent. We say that it is* non-trivial *if the message space has more than one element.*

We make a few observations about the definition of COA security. It does not prevent accept from accepting invalid secret-keys (i.e., those never produced by skGen). However, existential consistency guarantees that such secret-keys will behave "correctly." pkId and msgId provide a similar guarantee for maliciously crafted ciphertexts – that any ciphertext could be (existentially) interpreted *uniquely* (if at all) as obtained by encrypting a message using a valid public-key. It is also possible to have an accepted but invalid public-key (for which there is no secret-key that generates it which is accepted with non-negligible probability). However, ciphertexts produced by such a public-key will (almost) never be decrypted to anything other than $\perp$ by any accepted secret-key.

While the name COA security is quite broad, our definition above may appear quite limited in its goals. It just adds a deceptively simple set of consistency properties to a standard security definition. On the face of it, this definition may seem not to address various potential attacks that arise in a scenario with multiple keys. In particular, there is no strong correctness requirement associated with public keys that can get accepted by accept (unlike for secret keys). So an adversary may possibly send an invalid public-key which honest users may use to encrypt their messages, possibly resulting in the message being information-theoretically lost. However, a moment's reflection may reveal that this is not very different from a scenario where the adversary sends a valid public-key, but refuses to reveal the secret-key that generated it. Nevertheless, the reader may be left with an uneasy suspicion that our definition of COA security does not address all possible scenarios involving multiple secret-keys, public-keys and ciphertexts generated by honest and malicious parties interacting with each other.

The justification for the name comes from the fact that this definition suffices to achieve security in a general and abstract model of Cryptographic Agents in Section 5. Analyzing security in this abstract model requires a detailed (and tedious) argument, but the properties needed for meeting the strong security requirement in that model can be condensed to COA security. In fact, though we omit a formal statement or proof, any scheme secure in that model will *essentially* have to satisfy COA security.[13]

Before moving on to constructing a COA secure PKE scheme, we point out an implication of COA security that will be useful later in Section 6.

[13] A slight relaxation is possible in the definition of COA security, so that $\mathsf{pkId}(CT)$ can be $\perp$ if CT is unencryptable by any secret-key (even for CT generated by the encryption algorithm). For simplicity, we omit this relaxation. The equivalence with the definition from the Cryptographic Agents framework holds with this relaxation, and in the standard model which uses a *non-uniform* adversary model.

If the agents framework is instantiated for uniform adversaries or in the random oracle model, then correspondingly COA security can be further relaxed. The relaxed consistency requirement can be formulated in terms of an online game between a uniform PPT adversary and a computationally unbounded "interpreter" which assigns relationships

Ciphertext Resistance. Ciphertext resistance requires that any PPT adversary who is given *oracle access* to the encryption and decryption algorithms with an honestly generated key pair (but not the keys themselves) has negligible probability of generating a *new* valid ciphertext for this secret key (i.e., a ciphertext that is different from the ones returned by the encryption oracle, which on decryption using the secret key yields a non-$\perp$ outcome). In Appendix G we show that this follows from COA-security:

Lemma 1. *Any non-trivial COA-secure encryption scheme is ciphertext-resistant.*

4 Constructing COA Secure PKE

COA security imposes a fairly natural additional requirement on a Anon-CCA secure encryption scheme. We show that a couple of simple modifications can transform any Anon-CCA secure PKE scheme into a COA secure PKE scheme, if the former satisfies a simpler and natural correctness property that we refer to as *universal key reliability.* Later we shall show that this extra requirement can be removed, and hence a COA secure encryption scheme can be based on *any* Anon-CCA secure encryption.

4.1 From Anon-CCA Secure PKE with Universal Key Reliability

Let $\mathsf{pke} = (\mathsf{pke.skGen}, \mathsf{pke.pkGen}, \mathsf{pke.encrypt}, \mathsf{pke.decrypt})$ be a Anon-CCA-secure public-key encryption scheme with the following correctness property:

> *Universal Key Reliability:* For all SK that can be produced by $\mathsf{pke.skGen}$ with positive probability, for any message m in the message-space, $\Pr[\mathsf{pke.decrypt}(\mathit{SK}, \mathsf{pke.encrypt}(\mathit{PK}, m)) \neq m]$ is negligible, where $\mathit{PK} = \mathsf{pke.pkGen}(\mathit{SK})$ (the probability being over the randomness of $\mathsf{pke.encrypt}$).

Note that this is a relatively mild correctness requirement, as it needs to hold only for secret-keys that can actually be generated by $\mathsf{pke.skGen}$, and also needs to hold only with high probability for honestly generated ciphertexts. In particular, it does not rule out the possibility that a (maliciously crafted) ciphertext could be decrypted into different messages by different secret-keys corresponding to a public-key. However, universal key reliability does go beyond the basic correctness guarantee for PKE, which requires the error probability to be negligible only when averaged also over the choice of the secret-key (i.e., there could be bad secret-keys which can behave arbitrarily, as long as they are unlikely to be produced). In the second construction below we shall remove this requirement on pke, but given that this is a natural correctness property that holds for typical encryption schemes used in practice, we describe a relatively efficient modification that can make such schemes COA secure.

Firstly, we apply a simple modification which treats the random-tape for $\mathsf{pke.skGen}$ as the secret-key. This has the advantage that we need not check whether a given secret-key could be valid, as all random-tapes (padded up with 0's if necessary) are valid. This modification will let us extend the above correctness guarantee (stated for all secret-keys that can be generated by $\mathsf{pke.skGen}$) to all secret-keys. Given this, the heart of our modification is to ensure that each ciphertext can be generated by at most one public-key (this lets us define pkId, at least for honestly generated ciphertexts) and all the secret-keys leading to a public-key "behave the same way" (this will let us define msgId via decryption using a secret-key identified this way).

To ensure that all secret-keys that generate the same public-key decrypt identically, we shall simply include a commitment to the secret-key (or rather, the part of the secret-key that corresponds to the secret-key from the underlying PKE scheme pke) in our public-key. This will leave us with the problem of ensuring a well-defined pkId.

As a naïve attempt towards defining pkId, consider concatenating the public-key to the ciphertext: i.e., setting our new ciphertext as

$$\mathit{CT}^\star = (\mathit{PK}, \mathsf{pke.encrypt}(\mathit{PK}, m)).$$

among objects in a consistent manner; even if "collisions" may exist, as long as the adversary cannot discover them (adversary being uniform, or collisions being hidden by the random oracle), consistency may be maintained by the interpreter. For the sake of brevity, we omit a detailed presentation of this definition.

Clearly this will not be key-anonymous, but it would let us define pkId as required. To restore key-anonymity one may try to move the public-key inside the encryption, as $\mathsf{CT}^\star = \mathsf{pke.encrypt}(\mathsf{PK}, \mathsf{PK}||m)$. While this does recover the $\mathsf{Anon\text{-}CCA}$ security property, it no more lets us define a pkId function. In particular, it remains possible that there are two distinct key pairs $(\mathsf{PK}_1, \mathsf{SK}_1)$ and $(\mathsf{PK}_2, \mathsf{SK}_2)$ and distinct messages m, m' such that $\mathsf{pke.encrypt}(\mathsf{PK}_1, \mathsf{PK}_1||m)$ and $\mathsf{pke.encrypt}(\mathsf{PK}_2, \mathsf{PK}_2||m')$ are identically distributed. Instead, we may consider letting

$$\mathsf{CT}^\star = (\mathsf{c}, \mathsf{pke.encrypt}(\mathsf{PK}, m)),$$

where c is obtained using a perfectly binding commitment scheme com as $(\mathsf{c}, \mathsf{d}) \leftarrow \mathsf{com.Commit}(\mathsf{PK})$. However, this breaks CCA security, as the adversary can replace the first component with a fresh commitment to PK to obtain a valid ciphertext. Further, a maliciously crafted ciphertext in which the first component does not match PK will still be decrypted the same way as a valid ciphertext, and hence the relation between decryption and pkId essentially requires that pkId ignores the first component, leaving us with the same problems in defining pkId as before. In order to fix these issues, one may include the decommitment information along with the message being encrypted: i.e.,

$$\mathsf{CT}^\star = (\mathsf{c}, \mathsf{pke.encrypt}(\mathsf{PK}, \mathsf{d}||m)),$$

where $(\mathsf{c}, \mathsf{d}) \leftarrow \mathsf{com.Commit}(\mathsf{PK})$. This does not yet guarantee CCA security, as it leaves open the possibility of modifying the commitment without changing the decommitment information. Hence, instead of $\mathsf{d}||m$ above we shall use $\mathsf{c}||\mathsf{d}||m$ (unless $\mathsf{d}||m$ can be used to efficiently and uniquely compute c). Finally, note that for pkId, we need to uniquely associate a public-key for our scheme (and not for the given scheme pke), and hence we actually need $(\mathsf{c}, \mathsf{d}) \leftarrow \mathsf{com.Commit}(\mathsf{PK}^\star)$, where $\mathsf{PK}^\star$ is of the form $(\mathsf{c}', \mathsf{PK})$ with $(\mathsf{c}', \mathsf{d}') \leftarrow \mathsf{com.Commit}(\mathsf{SK})$. Incidentally, the additional randomness in $\mathsf{PK}^\star$ (beyond that of PK) is crucial for arguing $\mathsf{Anon\text{-}CCA}$ of our construction. The above scheme is summarized in Figure 8. In Appendix E we argue that it satisfies COA security.

4.2 From any Anon-CCA secure scheme

If the underlying PKE scheme does not offer universal key reliability, we need to design our scheme to achieve a similar property (the last item in existential consistency). The key to enforcing this property is to carry out a probabilistic check on the secret-key before accepting it. Roughly, a probabilistic check can be used to ensure that with good probability secret-key will decrypt encryption of *random messages* correctly. Then a randomization of the messages to encrypt and an error-correction step during decryption can be used to ensure that honest encryption using *every secret key that is accepted* will result in correct decryption, except with negligible probability over the randomness of encryption and randomness in the acceptance test.

However, one needs to be careful to preserve $\mathsf{Anon\text{-}CCA}$ security while modifying a given $\mathsf{Anon\text{-}CCA}$ encryption scheme in this manner. In particular, including error-correction provides the adversary with an easy way to corrupt a ciphertext and still have it decrypted. Our construction will use a combination of (strong, one-time) signatures and the techniques from the previous construction, along with the randomization/error-correction discussed above. The use of secret-sharing introduces a new avenue for attack, which is thwarted by appending to the shares tags which are included in the public-key. The full construction is presented in Figure 1. The proof of security, presented in Appendix F, proves Theorem 1.

4.3 Practical COA Secure Schemes

We point out that COA-security can be achieved with a relatively low overhead, and hence it would be fairly practical to require PKE schemes used in practice to meet this extra security requirement.

Given a Anon-CCA secure PKE scheme, $\mathsf{pke} = (\mathsf{pke.skGen}, \mathsf{pke.pkGen}, \mathsf{pke.encrypt}, \mathsf{pke.decrypt})$ a perfectly binding commitment scheme com, and a strong existentially unforgeable one-time signature scheme Sign, constructing a PKE scheme $\mathsf{pke}^\star$.
In this construction we consider a finite message-space (which can be extended to an arbitrary message space using standard hybrid encryption). For concreteness, let the message-space be $\mathcal{M} = \{0,1\}^\ell$ and $\mathsf{length}(m) = \ell$ where ℓ can be a function of κ. The given scheme pke is assumed to admit the message-space $\{0,1\}^{\ell+\kappa}$. We use a parameter $t = \omega(\log \kappa)$ (say $t = \log^2 \kappa$). Objects defined as tuples are unambiguously encoded into strings, and all the objects include implicit indicators as to which algorithms produced them.

- $\mathsf{pke}^\star.\mathsf{skGen}$. It outputs $(r_{\mathsf{pke}}, r_{\mathsf{com}}, \{r_0^i, r_1^i\}_{i\in[t]})$, where r_{pke} is a freshly sampled random-tape for pke.skGen and r_{com} is a freshly sampled random-tape used by com.Commit to commit to an element in the secret-key space $\mathcal{SK}$, and $r_b^i \leftarrow \{0,1\}^\kappa$.
- $\mathsf{pke}^\star.\mathsf{pkGen}(\mathit{SK}^\star)$. Parse $\mathit{SK}^\star$ as $(r_{\mathsf{pke}}, r_{\mathsf{com}}, \{r_0^i, r_1^i\}_{i\in[t]})$. Let $\mathit{SK} \leftarrow \mathsf{pke.skGen}$ using random-tape r_{pke}. Compute $\mathit{PK} := \mathsf{pke.pkGen}(\mathit{SK})$ and $(c, d) := \mathsf{com.Commit}(\mathit{SK}; r_{\mathsf{com}})$. Output $(\mathit{PK}, c, \{r_0^i, r_1^i\}_{i\in[t]})$. Output $\bot_{\text{PK}}$ if any of the intermediate steps (including parsing the input) fails or outputs $\bot$.
- $\mathsf{pke}^\star.\mathsf{encrypt}(\mathit{PK}^\star, m)$. Let $(c^\star, d^\star) \leftarrow \mathsf{com.Commit}(\mathit{PK}^\star)$ and $(\mathit{sigk}, \mathit{verk}) \leftarrow \mathsf{sig.keyGen}$. Parse $\mathit{PK}^\star$ as $(\mathit{PK}, c, \{r_0^i, r_1^i\}_{i\in[t]})$. For each $i \in [t]$, additively secret-share $(m, d^\star, c^\star, \mathit{verk})$ into a pair (m_0^i, m_1^i), and define $\mu_b^i = r_b^i || m_b^i$ and $\gamma = \{\mathsf{pke.encrypt}(\mathit{PK}, \mu_b^i)\}_{i\in[t], b\in\{0,1\}}$. ($\gamma = \bot$ if any of the steps above fails.) Let $\xi = (c^\star, \gamma)$ and let $\tau \leftarrow \mathsf{sig.Sign}(\mathit{sigk}, \xi)$. Output (ξ, τ) as the ciphertext.
- $\mathsf{pke}^\star.\mathsf{decrypt}(\mathit{SK}^\star, \mathit{CT}^\star)$. Parse $\mathit{CT}^\star$ as (ξ, τ), and further parse ξ as $(c^\star, \{\mathit{CT}_0^i, \mathit{CT}_1^i\}_{i\in[t]})$. Parse $\mathit{SK}^\star$ as $(r_{\mathsf{pke}}, r_{\mathsf{com}}, \{r_0^i, r_1^i\}_{i\in[t]})$. Then do the following:
  1. Compute $\mathit{PK}^\star := \mathsf{pke}^\star.\mathsf{pkGen}(\mathit{SK}^\star)$. Along the way, this obtains SK by running pke.skGen with random-tape r_{pke}.
  2. Let $\mu_b^i = \mathsf{pke.decrypt}(\mathit{SK}, \mathit{CT}_b^i)$. Parse μ_b^i as $s_b^i || m_b^i$, where $s_b^i \in \{0,1\}^\ell$.
  3. Check if there is a set $S \subseteq [t]$, $|S| > t/2$ such that $\exists m, d^\star, \mathit{verk}$, $\forall i \in S$, $s_0^i = r_0^i$, $s_1^i = r_1^i$ and $m_0^i \oplus m_1^i = (m, d^\star, c^\star, \mathit{verk})$.
  4. If so, check if $\mathit{PK}^\star = \mathsf{com.Open}(c^\star, d^\star)$. and $\mathsf{sig.Verify}(\mathit{verk}, \xi, \tau) = 1$.
  5. If all the checks pass, output m. If any of the steps fail, output $\bot$.
- $\mathsf{pke}^\star.\mathsf{accept}(\mathit{obj})$. If obj has the form of a public-key (including $\bot_{\text{PK}}$) or ciphertext above, $\mathsf{pke}^\star.\mathsf{accept}(\mathit{obj})$ outputs PK or CT respectively. However, if obj has the form of a secret-key, it proceeds as follows: Parse obj as $(r_{\mathsf{pke}}, r_{\mathsf{com}}, \{r_0^i, r_1^i\}_{i\in[t]})$, and compute $\mathit{SK} \leftarrow \mathsf{pke.skGen}$ using random-tape r_{pke}. Pick κ random strings $\rho_i \in \{0,1\}^\ell$ and for $i \in [\kappa]$, check if $\mathsf{pke.decrypt}(\mathit{SK}, \mathsf{pke.encrypt}(\mathit{PK}, \rho_i)) = \rho_i$; output SK if all the checks pass and output $\bot$ otherwise. (Output $\bot$ if obj does not have a form that matches a valid object.)

Fig. 1 A COA secure PKE scheme without assuming universal key reliability for the underlying PKE scheme.

Hybrid Encryption. It is easy to see that hybrid encryption, by combining with a CCA secure symmetric-key encryption scheme, preserves COA security.[14] Note that in such a hybrid scheme, the overhead of COA security applies only to short messages (keys), independent of the actual size of the data being encrypted.
Existing PKE Schemes. The Cramer-Shoup encryption scheme [14], with a minor modification, satisfies COA security. The modification, which was proposed by Abdalla et al. [1], simply makes a pathological ciphertext that is independent of the public-key to be invalid. It was shown in [1] that this leaves the scheme Anon-CCA secure (and also makes it meet a robustness property). For naturally defined algorithms accept, pkId and msgId (with accept requiring efficient recognizability of the group elements), it can be verified that this scheme satisfies existential consistency too.
Relaxing COA Security. As observed before (see Footnote 13) COA security can be relaxed by restricting to uniform adversaries, or by allowing a random-oracle. Then, the commitment can be replaced by a computationally binding commitment or even a random-oracle-based commitment.

[14] A standard hybrid argument establishes that the resulting scheme is Anon-CCA secure. Existential consistency follows by considering the same pkId as in the given scheme (applied to the PKE part of the ciphertext), and defining a new msgId function which uses the original msgId function to obtain a key for the SKE and then uses it to decrypt the SKE part of ciphertext.

By the above observations, combined with the fact that many of the practical PKE schemes satisfy the mild universal key reliability condition (allowing the first transformation above to be used), obtaining COA security has a very low overhead in terms of computation and communication.

5 Extending Cryptographic Agents

As discussed in Section 1, the framework of [2] does not suffice for our purposes. Specifically, we need to extend the framework so that the user can transfer agents to the test. Further, it should be possible (for the user and the test) to upload an agent and get private access to it, i.e., unless the test (or the user) explicitly asks an for agent to be transferred, it will not be.

We allow adversaries to transfer objects to tests in much the same way as tests transfer to them. Thus, both these worlds are symmetric with respect to how test and user use the schema, and how they communicate with each other. We permit maliciously created keys as objects in the system and allow these to be transferred like any other object. Additionally, we allow multiple keys at a given time and capture interaction between arbitrary mismatched objects.[15]

We proceed to describe the model formally.

5.1 The Model

The formalization of the agents framework makes use of interactive Turing machines (ITM) with tapes for input, output, incoming communication, outgoing communication, randomness and work-space. A *schema* is analogous to functionality in the Universal Composition framework for secure multiparty computation, and specifies what is legitimate for a user to do in a system. A schema defines the families of agents that a "user" and a "test" are allowed to create. Agents can interact with one another in a *session*: the first agent is executed till it enters a blocking or halting state, and then the second and so forth, in a round-robin fashion, until all the agents remain in blocking or halting states for a full round. Please see Appendix C for details.

Ideal World Model. A schema $\boldsymbol{\Sigma}$ is simply a family of agents.[16] The ideal system for a schema $\boldsymbol{\Sigma}$ consists of two parties Test and User and a fixed third party $\mathcal{B}[\boldsymbol{\Sigma}]$ (for "black-box"). All three parties are probabilistic polynomial time (PPT) ITMs, have a security parameter κ built-in. Test and User may be non-uniform. Test receives a *test-bit* b as input and User produces an output bit b'.

$\mathcal{B}[\boldsymbol{\Sigma}]$ maintains two lists of handles $\mathsf{R}^{\mathsf{Test}}$ and $\mathsf{R}^{\mathsf{User}}$, which contain the set of handles belonging to Test and User respectively. Each handle in these lists is mapped to an agent. At the beginning of an execution, both the lists are empty. While Test and User can arbitrarily talk to each other, the interaction with $\mathcal{B}[\boldsymbol{\Sigma}]$ can be summarized as follows:

– **Creating agents.** Test and User can, at any point, choose an agent from $\boldsymbol{\Sigma}$ and send it to $\mathcal{B}[\boldsymbol{\Sigma}]$ for instantiation. More precisely, they can send a command $(\mathsf{init}, P, \mathsf{str})$ to $\mathcal{B}[\boldsymbol{\Sigma}]$, where $P \in \boldsymbol{\Sigma}$ and str is an initial input for the agent. Then, $\mathcal{B}[\boldsymbol{\Sigma}]$ will instantiate the agent (with an empty work-tape) and run it with str and security parameter as inputs. It then stores $(h, \mathtt{config})$ in the list of the party who sent the command ($\mathsf{R}^{\mathsf{Test}}$ or $\mathsf{R}^{\mathsf{User}}$) where $\mathtt{config}$ is the agent's configuration after the execution and h is a new handle (say, simply, the number of handles stored so far in the list); h is returned to the relevant party (Test or User).

[15] Among other things, this simplifies the original formulation by not requiring separate agent families for tests and users. In the original formulation, when only one key was allowed in the system, separate agent families were used to model the fact that in the ideal world only the test should do certain privileged operations (like decryption, or function key-generation in functional encryption). This is now modeled by the ability of the test to generate a secret key object and not transfer it to the user.

[16] Typically, this family will have a single uniform agent, for all security parameters.

- **Request for Session Execution.** At any point in time, Test or User may request an execution of a session. We describe the process when Test requests a session execution; the process for User is symmetric. Test can send a command $(\mathsf{run}, (h_1, x_1) \ldots, (h_t, x_t))$, where h_i are handles obtained in the list $\mathsf{R}^{\mathsf{Test}}$, and x_i are input strings for the corresponding agents. [17] $\mathcal{B}[\boldsymbol{\Sigma}]$ executes a session with the agents with starting configurations in $\mathsf{R}^{\mathsf{Test}}$, corresponding to the specified handles, with their respective inputs, till it terminates. It obtains a collection of outputs $(y_1, \ldots, y_t)$ and updated configurations of agents. It generates new handles $h'_1, \ldots, h'_t$ corresponding to the updated configurations, adds them to $\mathsf{R}^{\mathsf{Test}}$, and returns $(h'_1, \ldots, h'_t, y_1, \ldots, y_t)$ to Test. (If an agent halts in a session, no new handle is given out for that agent).
- **Transferring agents.** Test can send a command $(\mathsf{transfer}, h)$ to $\mathcal{B}[\boldsymbol{\Sigma}]$ upon which it looks up the entry $(h, \texttt{config})$ from $\mathsf{R}^{\mathsf{Test}}$ (if such an entry exists) and adds an entry $(h', \texttt{config})$ to $\mathsf{R}^{\mathsf{User}}$, where h' is a new handle, and sends the handle h' to User. Symmetrically, User can transfer an agent to Test using the transfer command.

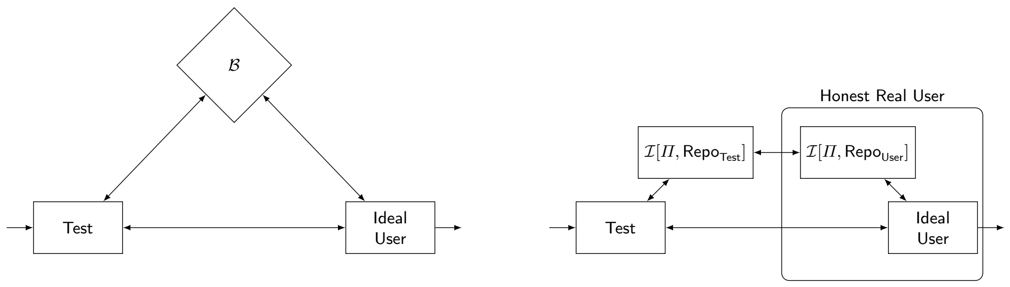
We define the random variable $\text{IDEAL}\langle \mathsf{Test}(b) \mid \boldsymbol{\Sigma} \mid \mathsf{User} \rangle$ to be the output of User in an execution of the above system, when Test gets b as the test-bit. We write $\text{IDEAL}\langle \mathsf{Test} \mid \boldsymbol{\Sigma} \mid \mathsf{User} \rangle$ to denote the output when the test-bit is a uniformly random bit. We also define $\text{TIME}\langle \mathsf{Test} \mid \boldsymbol{\Sigma} \mid \mathsf{User} \rangle$ as the maximum number of steps taken by Test (with a random input), $\mathcal{B}[\boldsymbol{\Sigma}]$ and User in total.

In this work, we use the notion of *statistical* hiding in the ideal world as introduced in [3], rather than the original notion used in [2]. (This still results in a security definition that subsumes the traditional definitions, as they involve tests that are statistically hiding.)

Definition 3 ((Statistical) Ideal world hiding). *A* Test *is* s-hiding w.r.t. a schema $\boldsymbol{\Sigma}$ *if, for all unbounded users* User *who make at most a polynomial number of queries,*

$$\text{IDEAL}\langle \mathsf{Test}(0) \mid \boldsymbol{\Sigma} \mid \mathsf{User} \rangle \approx \text{IDEAL}\langle \mathsf{Test}(1) \mid \boldsymbol{\Sigma} \mid \mathsf{User} \rangle.$$

When the schema is understood, we shall abbreviate the property of being "s-hiding w.r.t. a schema" as simply being "ideal-hiding."



$\mathcal{B}$
Test
Ideal User
Honest Real User
$\mathcal{I}[\Pi, \mathsf{Repo}_{\mathsf{Test}}]$
$\mathcal{I}[\Pi, \mathsf{Repo}_{\mathsf{User}}]$
Test
Ideal User

Fig. 2 The ideal world (on the left) and the real world with an honest user.

Real World Model The real world for a schema $\boldsymbol{\Sigma}$ consists of two parties Test and User that interact with each other arbitrarily, as in the ideal world. However, the third party $\mathcal{B}[\boldsymbol{\Sigma}]$ in the ideal world is replaced by two other parties $\mathcal{I}[\Pi, \mathsf{Repo}_{\mathsf{Test}}]$ and $\mathcal{I}[\Pi, \mathsf{Repo}_{\mathsf{User}}]$ (when User is honest), which run the algorithms specified by a *cryptographic scheme* Π. A cryptographic scheme (or simply scheme) Π is a collection of stateless (possibly

[17] Note that if the same handle appears more than once in the tuple $(h_1, \ldots, h_t)$, it is interpreted as multiple agents with the same configuration (but possibly different inputs). Also note that after a session, the old handles for the agents are not invalidated; so a party can access a configuration of an agent any number of times, by using the same handle.

randomized) algorithms $\mathit{\Pi}.\mathsf{init}, \mathit{\Pi}.\mathsf{run}$ and $\mathit{\Pi}.\mathsf{receive}$, which use a repository Repo to store a mapping from handles to objects. More precisely, the repository is a table with entries of the form $(\mathsf{h}, \mathsf{obj})$, where h is a unique handle (say, a non-negative integer) and obj is a cryptographic object (represented, for instance, as a binary string). At the start of an execution, Repo is empty.

If a scheme implementation ($\mathcal{I}[\mathit{\Pi}, \mathsf{Repo}_{\mathsf{Test}}]$ or $\mathcal{I}[\mathit{\Pi}, \mathsf{Repo}_{\mathsf{User}}]$) receives input $(\mathsf{init}, P, \mathsf{str})$, then it runs $\mathit{\Pi}.\mathsf{init}(P, \mathsf{str})$ to obtain an object obj which is added to Repo and a handle is returned. If it receives the command $(\mathsf{run}, (\mathsf{h}_1, x_1), \cdots, (\mathsf{h}_t, x_t))$, then objects $(\mathsf{obj}_1, \ldots, \mathsf{obj}_t)$ corresponding to $(\mathsf{h}_1, \ldots, \mathsf{h}_t)$ are retrieved from Repo and $\mathit{\Pi}.\mathsf{run}((\mathsf{obj}_1, x_1), \ldots, (\mathsf{obj}_t, x_t))$ is evaluated to obtain $((\mathsf{obj}'_1, y_1), \ldots, (\mathsf{obj}'_t, y_t))$ where obj'_i are new objects and y_i are output strings; the objects are added to Repo, with a new handle for each, and the new handles, along with the outputs, are returned. (If an obj'_i is empty, then no new handle is added; this corresponds to an agent having halted.)

$\mathcal{I}[\mathit{\Pi}, \mathsf{Repo}_{\mathsf{Test}}]$ and $\mathcal{I}[\mathit{\Pi}, \mathsf{Repo}_{\mathsf{User}}]$ do not interact with each other, except when one of them receives a $\mathsf{transfer}$ command. If Test sends a command $(\mathsf{transfer}, \mathsf{h})$ to $\mathcal{I}[\mathit{\Pi}, \mathsf{Repo}_{\mathsf{Test}}]$, it looks for an entry $(\mathsf{h}, \mathsf{obj})$ in $\mathsf{Repo}_{\mathsf{Test}}$ and sends obj to $\mathcal{I}[\mathit{\Pi}, \mathsf{Repo}_{\mathsf{User}}]$; on receiving obj from $\mathcal{I}[\mathit{\Pi}, \mathsf{Repo}_{\mathsf{Test}}]$, $\mathcal{I}[\mathit{\Pi}, \mathsf{Repo}_{\mathsf{User}}]$ will run $\mathit{\Pi}.\mathsf{receive}(\mathsf{obj})$ which outputs (a possibly modified) object obj' and if $\mathsf{obj}' \neq \perp$, $\mathcal{I}[\mathit{\Pi}, \mathsf{Repo}_{\mathsf{User}}]$ will add $(\mathsf{h}', \mathsf{obj}')$ to $\mathsf{Repo}_{\mathsf{User}}$, where h' is a new handle, and outputs h' to User. The process of User transferring an object to Test is symmetric.

The purpose of the $\mathsf{receive}$ algorithm is as follows. When an object is transferred to $\mathcal{I}[\mathit{\Pi}, \mathsf{Repo}_{\mathsf{User}}]$, it may be required to perform some tests to check whether the received object must be rejected. These checks are performed only a single time when the object is transferred, and once the object is accepted as valid, it must not be tested again. This is to prevent the scenario that an object is tested each time it is used: aside from the inefficiency of repeating this operation, note that the checks may be probabilistic and may pass sometimes and fail at other times. Since this is not captured in the ideal world, an object is tested and received once and for all.

Note that we *do not* allow Test direct access to the cryptographic objects stored in its repository. In particular, it cannot send a handle to $\mathsf{Repo}_{\mathsf{Test}}$, and get the object corresponding to it in return. Also observe that if User is corrupt, which we denote by Adv, it may not run the scheme it is supposed to. It can run any arbitrary algorithm and send any object of its choice to $\mathcal{I}[\mathit{\Pi}, \mathsf{Repo}_{\mathsf{Test}}]$.

We define the random variable $\textsc{real}\langle \mathsf{Test}(b) \mid \mathit{\Pi} \mid \mathsf{Adv} \rangle$ to be the output of Adv in an execution of the above system involving Test with test-bit b, $\mathcal{I}[\mathit{\Pi}, \mathsf{Repo}_{\mathsf{User}}]$ and Adv; as before, we omit b from the notation to indicate a random bit. Also, as before, $\textsc{Time}\langle \mathsf{Test} \mid \mathit{\Pi} \mid \mathsf{Adv} \rangle$ is the maximum number of steps taken by Test (with a random input), $\mathcal{I}[\mathit{\Pi}, \mathsf{Repo}_{\mathsf{User}}]$ and Adv in total.

Definition 4. *We say that* Test *is* hiding w.r.t. $\mathit{\Pi}$ *if* $\forall$ *PPT party* Adv*,*

$$\textsc{real}\langle \mathsf{Test}(0) \mid \mathit{\Pi} \mid \mathsf{Adv} \rangle \approx \textsc{real}\langle \mathsf{Test}(1) \mid \mathit{\Pi} \mid \mathsf{Adv} \rangle.$$

When $\mathit{\Pi}$ is understood, we may simply say that Test is real-hiding.

5.2 Security Definition

We are ready to present the security definition of a cryptographic agent scheme $\mathit{\Pi}$ implementing a schema $\boldsymbol{\Sigma}$. Below, the *honest real-world user*, corresponding to an ideal-world user User, is defined as the composite program $\mathcal{I}[\mathit{\Pi}, \mathsf{Repo}_{\mathsf{User}}] \circ \mathsf{User}$ as shown in Figure 2.

Let $\mathit{\Gamma}_{\mathsf{ppt}}$ denote the family of all PPT Test.

Definition 5. *A cryptographic agent scheme $\mathit{\Pi}$ is said to be a $\mathit{\Gamma}$-s-*IND-PRE-secure scheme *for a schema $\boldsymbol{\Sigma}$ if the following conditions hold.*

- *Correctness.* $\forall$ PPT User *and* $\forall$ $\mathsf{Test} \in \mathit{\Gamma}_{\mathsf{ppt}}$, $\textsc{ideal}\langle \mathsf{Test} \mid \boldsymbol{\Sigma} \mid \mathsf{User} \rangle \approx \textsc{real}\langle \mathsf{Test} \mid \mathit{\Pi} \mid \mathcal{I}[\mathit{\Pi}, \mathsf{Repo}_{\mathsf{User}}] \circ \mathsf{User} \rangle$*. If equality holds, $\mathit{\Pi}$ is said to have perfect correctness.*
- *Efficiency. There exists a polynomial* poly *such that,* $\forall$ PPT User*,* $\forall$ $\mathsf{Test} \in \mathit{\Gamma}_{\mathsf{ppt}}$*,* $\textsc{Time}\langle \mathsf{Test} \mid \mathit{\Pi} \mid \mathcal{I}[\mathit{\Pi}, \mathsf{Repo}_{\mathsf{User}}] \circ \mathsf{User} \rangle \leq \mathrm{poly}(\textsc{Time}\langle \mathsf{Test} \mid \boldsymbol{\Sigma} \mid \mathsf{User} \rangle, \kappa)$.

– *(Statistical) Indistinguishability Preservation.* $\forall\, \mathsf{Test} \in \Gamma$*,* Test *is s-hiding w.r.t.* $\boldsymbol{\Sigma} \Rightarrow \mathsf{Test}$ *is hiding w.r.t.* Π*.*

When Γ *is the family* Γ_{ppt}*, we simply say that* Π *is an* IND-PRE*-secure scheme for* $\boldsymbol{\Sigma}$*.*

Note that the correctness and efficiency requirements are w.r.t. all PPT Test.

Δ ***test-family.*** We say that $\mathsf{Test} \in \Delta$, if it behaves as follows: every init and run command it sends to $\mathcal{B}[\boldsymbol{\Sigma}]$ is sent to User. For transfer commands, it picks two handles h_0, h_1 and sends a message $(\mathsf{transfer}, h_0, h_1)$ to User and sends $\mathsf{transfer}[h_b]$ to $\mathcal{B}[\boldsymbol{\Sigma}]$, where b is the test-bit. In the sequel we shall focus on Δ-s-IND-PRE security which is defined by invoking Definition 5 with $\Gamma = \Delta$.

6 Modeling and Implementing Public-key encryption in Agents Framework

The Schema $\boldsymbol{\Sigma}_{\mathsf{pke}}$*.* We formalize PKE as a schema $\boldsymbol{\Sigma}_{\mathsf{pke}}$ shown in Figure 3. This schema presents a safe, but flexible, interface for PKE. For instance, unlike a UC-security functionality, the schema allows keys that can be transferred. But this also presents new security issues that need to be addressed.

The interface includes the usual commands like `encr` and `decr` that correspond to encryption and decryption, but also other ones like `type` (to check if an agent is a ciphertext, public-key or secret-key) and `compare` (to check if two agents are the same). Key generation is modeled in two steps – an agent initialization, to set up a secret key, and `pkGen` to derive a public key from the secret key. We also allow a command `clone` that allows one to "clone" a secret key into a different one which has the same public key.[18] Every component of an encryption scheme — secret-key, public-key, and ciphertexts — are treated as objects that can be maliciously generated and distributed.

A Scheme Π_{pke} *Implementing* $\boldsymbol{\Sigma}_{\mathsf{pke}}$*.* In Figure 4 we give an implementation Π_{pke} of the schema $\boldsymbol{\Sigma}_{\mathsf{pke}}$ defined above using a COA-secure PKE scheme $\mathsf{pke}^{\star}$. The implementation itself is fairly intuitive and natural. Recall that a scheme implementing a schema offers the same interface as the schema itself, and should have three procedures, init, run and receive. In particular, run offers the same interface as the run command of the schema, but with handles replaced by "objects" (strings): i.e., the input and output of a run session consists of pairs of the form (obj, x), where obj is an object (an updated object in the case of output) and x is the input or output associated with that object.

7 Sketch of Proof of Security of $\boldsymbol{\Pi}_{\mathsf{pke}}$

In this section we sketch the proof of Theorem 2, which involves showing that Π_{pke} securely implements $\boldsymbol{\Sigma}_{\mathsf{pke}}$, assuming that $\mathsf{pke}^{\star}$ is a COA-secure PKE scheme. (The full proof appears in Appendix I.) A key element in carrying this out is the notion of an *extended schema*, that will be used to get intermediate security guarantees. Before presenting the proof sketch we introduce this formalism.

7.1 Extended Schemas

In our standard model, the ideal world is *symmetric* with respect to Test and User: both Test and User interact with $\mathcal{B}[\boldsymbol{\Sigma}]$ in the same way. However, as a useful intermediate tool, we shall employ the notion of an *extended schema*, where this symmetry is broken. In an extended schema $(\boldsymbol{\Sigma}, \boldsymbol{\Sigma}^{\dagger})$, $\boldsymbol{\Sigma}^{\dagger}$ is an additional family of agents which can be instantiated (using init) only by a corrupt User. Test, as well as an honest User, is restricted to instantiating agents in $\boldsymbol{\Sigma}$. Extended schemas can be used to capture non-ideal features of a schema.

We shall use an extended schema within our proof of security, as a means to first reason about the security of secret-keys created by honest users only; at this point we shall not have any particular security guarantees

[18] Even though honest users may not require a cloning facility, our implementations will provide it to the adversary. To remove the need for an implementation to provide this facility to the honest users as well, we can consider an extended schema $(\boldsymbol{\Sigma}^{-}_{\mathsf{pke}}, \boldsymbol{\Sigma}_{\mathsf{pke}})$ where $\boldsymbol{\Sigma}^{-}_{\mathsf{pke}}$ does not allow cloning keys. For simplicity, we avoid this formalism.

$\boldsymbol{\Sigma}_{\mathsf{pke}}$ consists of a single agent $\mathcal{P}_{\mathsf{pke}}$ which behaves as follows, when invoked in a session.[a]

1. **Initializing (as a secret-key).** When run with an empty work-tape (when the init command is sent to $\mathcal{B}[\boldsymbol{\Sigma}_{\mathsf{pke}}]$) and with the security parameter κ as input, the agent records a tuple $(\mathtt{sk}, \textsf{\textit{sk-tag}}, \textsf{\textit{pk-tag}})$ on its work-tape, where $\mathtt{sk}$ is a keyword, and $\textsf{\textit{sk-tag}}, \textsf{\textit{pk-tag}} \leftarrow_R \{0,1\}^\kappa$ are sampled uniformly at random.
2. **Generating Public Key from Secret key.** If the input is the keyword "$\mathtt{pkGen}$" and the work-tape has $(\mathtt{sk}, \textsf{\textit{sk-tag}}, \textsf{\textit{pk-tag}})$, then it changes the work-tape entry to $(\mathtt{pk}, \textsf{\textit{pk-tag}})$.[b]
3. **Encryption.** If the work-tape entry is $(\mathtt{pk}, \textsf{\textit{pk-tag}})$ and the input is the pair $(\mathtt{encr}, m)$, where $\mathtt{encr}$ is a keyword and m is a bit-string, then it updates the work-tape contents to $(\mathtt{ct}, m, \textsf{\textit{pk-tag}}, \textsf{\textit{ct-tag}})$, where $\textsf{\textit{ct-tag}} \leftarrow_R \{0,1\}^\kappa$ is freshly sampled.
4. **Decryption.** For decryption, it is expected that two agents – a secret-key agent and a ciphertext agent – will be invoked together in a session, each with the input keyword $\mathtt{decr}$. If an agent is invoked with this keyword as input, and its work-tape contents are $(\mathtt{ct}, m, \textsf{\textit{pk-tag}}, \textsf{\textit{ct-tag}})$ it sends $(\textsf{\textit{pk-tag}}, m)$ to the first agent in the session. If on the other hand, the input is $\mathtt{decr}$ and its work-tape contents are $(\mathtt{sk}, \textsf{\textit{sk-tag}}, \textsf{\textit{pk-tag}})$, it waits for a message from the second agent in the session; if it receives a message of the form $(\textsf{\textit{pk-tag}}^*, m)$ such that $\textsf{\textit{pk-tag}} = \textsf{\textit{pk-tag}}^*$, then it writes m on the output tape, otherwise it writes $\bot$.
5. **Recognizing the type of agent:** If an agent is invoked with the keyword $\mathtt{type}$ as input, it behaves as follows, depending on the contents of its work-tape:
   - if the work-tape has $(\mathtt{sk}, \textsf{\textit{sk-tag}}, \textsf{\textit{pk-tag}})$, output $\mathtt{sk}$.
   - if the work-tape has $(\mathtt{pk}, \textsf{\textit{pk-tag}})$, output $\mathtt{pk}$.
   - if the work-tape has $(\mathtt{ct}, m, \textsf{\textit{pk-tag}}, \textsf{\textit{ct-tag}})$, output $(\mathtt{ct}, \ell)$, where $\ell = \mathsf{length}(m)$.
6. **Comparing agents:** If an agent is invoked with the keyword $\mathtt{compare}$, it first sends the contents of its work-tape to the first agent in the session. Then, if it receives a message on its incoming communication tape from a single other agent, and it is *identical* to the contents on its own tape, it outputs true; else it outputs false. (If it receives no message, it outputs $\bot$.)
7. **Cloning a secret-key:** If the input is the keyword "$\mathtt{clone}$" and the work-tape has $(\mathtt{sk}, \textsf{\textit{sk-tag}}, \textsf{\textit{pk-tag}})$, then it samples $\textsf{\textit{sk-tag}}' \leftarrow_R \{0,1\}^\kappa$ and sets its work-tape contents to $(\mathtt{sk}, \textsf{\textit{sk-tag}}', \textsf{\textit{pk-tag}})$.

[a] In each item here, at the end the agent enters a blocking state, marking the end of the session, or when an agent does not update its work-tape, halts, to avoid creating a new handle unnecessarily. For brevity, we do not include these blocking and halting actions in the description of the agent.

[b] Note that if one runs a $\mathtt{sk}$-agent multiple times with the input $\mathtt{pkGen}$, we require that the resulting agent always has the same configuration.

Fig. 3 PKE-schema $\boldsymbol{\Sigma}_{\mathsf{pke}}$.

on the objects that do not correspond to the honest parties' secret-keys. In this intermediate situation, we idealize the objects derived from secret-keys generated by the honest parties and leave the other objects "real" – i.e., according to a scheme $\mathit{\Pi}$ being employed. This *semi-ideal* situation is formalized using the extended schema $(\boldsymbol{\Sigma}, \boldsymbol{\Sigma}^\dagger_{\mathit{\Pi}})$. $\boldsymbol{\Sigma}^\dagger_{\mathit{\Pi}}$ consists of a single agent $P^\dagger$, which provides the same interface as the the agents in $\boldsymbol{\Sigma}$, but internally works using the scheme $\mathit{\Pi}$. Specifically, when initialized with an object $\textsf{\textit{obj}}$ as input, $P^\dagger$ simply copies $\textsf{\textit{obj}}$ to its work-tape. Further, when a session is executed involving agents of this family, they simply execute $\mathit{\Pi}.\mathsf{run}$ on the objects recorded on their work-tapes, to derive new objects (which will be recorded on their work-tapes) and outputs.

While agents from each of $\boldsymbol{\Sigma}$ and $\boldsymbol{\Sigma}^\dagger_{\mathit{\Pi}}$ coexist in the system, their interaction with each other is restricted: if a session involving agents from both $\boldsymbol{\Sigma}$ and $\boldsymbol{\Sigma}^\dagger_{\mathit{\Pi}}$ is executed, all the agents would halt without output.

The extended schema we use in our proof is $\boldsymbol{\Sigma}_{\mathsf{ext}} = (\boldsymbol{\Sigma}_{\mathsf{pke}}, \boldsymbol{\Sigma}^\dagger_{\mathit{\Pi}_{\mathsf{pke}}})$, in which a corrupt user can create agents in the scheme $\boldsymbol{\Sigma}^\dagger_{\mathit{\Pi}_{\mathsf{pke}}}$ (as well as in $\boldsymbol{\Sigma}_{\mathsf{pke}}$), where $\mathit{\Pi}_{\mathsf{pke}}$, is the scheme whose security we are interested in proving.

Let $\mathsf{pke}^\star = (\mathsf{pke}^\star.\mathsf{skGen}, \mathsf{pke}^\star.\mathsf{pkGen}, \mathsf{pke}^\star.\mathsf{encrypt}, \mathsf{pke}^\star.\mathsf{decrypt}, \mathsf{pke}^\star.\mathsf{accept})$ be a COA-secure PKE scheme, with message space $\mathcal{M}$. The sets $\mathcal{SK}^\star$, $\mathcal{PK}^\star$ and $\mathcal{CT}^\star$ be as in Definition 1. Also, we will require that there is an efficiently computable function that reveals the message length from a ciphertext – i.e., a function $\mathsf{pke.msglength}$ such that for all $CT \in \mathcal{CT}^\star$, if for some $SK \in \mathcal{SK}^\star$, $m := \mathsf{decrypt}(SK, CT) \neq \perp$, then $\mathsf{pke.msglength}(CT) = \mathsf{length}(m)$.[a]

1. **Initialization.** $\Pi_{\mathsf{pke}}.\mathsf{init}()$ obtains $SK^\star \leftarrow \mathsf{pke}^\star.\mathsf{skGen}$, and $pad \leftarrow \{0,1\}^\kappa$ and outputs $(SK^\star || pad)$ as the intialized object.[b]
2. **Generating Public Key from Secret Key.** $\Pi_{\mathsf{pke}}.\mathsf{run}(obj, \mathtt{pkGen})$ sets $SK^\star$ to be all but the last κ bits of obj and outputs $(\mathsf{pke}^\star.\mathsf{pkGen}(SK^\star), \perp)$ (or $(\perp, \perp)$ if $|obj| < \kappa$).
3. **Encryption.** $\Pi_{\mathsf{pke}}.\mathsf{run}(obj, (\mathtt{encr}, m))$ outputs $(obj', \perp)$ where $obj' \leftarrow \mathsf{pke}^\star.\mathsf{encrypt}(obj, m)$.
4. **Decryption.** $\Pi_{\mathsf{pke}}.\mathsf{run}((obj_1, \mathtt{decr}), (obj_2, \mathtt{decr}))$ sets $SK^\star$ to be all but the last κ bits of obj_1 and outputs $((\perp, m), (\perp, \perp))$ where $m := \mathsf{pke}^\star.\mathsf{decrypt}(SK^\star, obj_2)$ (or $m := \perp$ if $|obj| < \kappa$).
5. **Recognizing the agent-type.** $\Pi_{\mathsf{pke}}.\mathsf{run}(obj, \mathtt{type})$ outputs $(\perp, \mathtt{sk})$ if it holds that $\mathsf{pke}^\star.\mathsf{accept}(SK^\star) = \textsc{sk}$, where $SK^\star$ is all but the last κ bits of obj. Otherwise, if $\mathsf{pke}^\star.\mathsf{accept}(obj) = \textsc{pk}$, it outputs $(\perp, \mathtt{pk})$. Otherwise, if $\mathsf{pke}^\star.\mathsf{accept}(obj) = \textsc{ct}$, it outputs $(\perp, (\mathtt{ct}, \ell))$, where $\ell := \mathsf{pke.msglength}(obj)$. If none of the above conditions holds, it outputs $(\perp, \perp)$.
6. **Comparing agents.** $\Pi_{\mathsf{pke}}.\mathsf{run}((obj_1, \mathtt{compare}), (obj_2, \mathtt{compare}))$ outputs $((\perp, \mathsf{true}), (\perp, \perp))$ if $obj_1 = obj_2$ and $((\perp, \mathsf{false}), (\perp, \perp))$ otherwise.
7. **Cloning Secret Keys.** $\Pi_{\mathsf{pke}}.\mathsf{run}(obj, \mathtt{clone})$ checks if $\Pi_{\mathsf{pke}}.\mathsf{run}(obj, \mathtt{type})$ (as defined above) yields $(\perp, \mathtt{sk})$ and if so outputs $(SK^\star || pad', \perp)$ where $SK^\star$ is all but the last κ bits of obj and $pad' \leftarrow \{0,1\}^\kappa$ is freshly sampled. Otherwise it outputs $(\perp, \perp)$.
8. **Receiving transferred agents.** $\Pi_{\mathsf{pke}}.\mathsf{receive}(obj)$ outputs $\perp$ if $\Pi_{\mathsf{pke}}.\mathsf{run}(obj, \mathtt{type})$ (as defined above) returns $(\perp, \perp)$, and otherwise outputs obj unchanged.

[a] If the honest users do not require the ability to learn $\mathsf{length}(m)$ from a ciphertext of m, this requirement can be removed; but this will result in a slightly more complex specification of the schema, in which the adversary has more capabilities than the honest users (cf. Footnote 18).

[b] pad is used to provide secret-key cloning as a feature. We remark that the ability to clone is not usually a useful feature for honest parties. As mentioned in Footnote 18, one can choose not to implement it, at the expense of a slightly more involved description of the schema which provides this feature only to the adversary.

Fig. 4 Implementing the PKE Schema.

7.2 Proof Sketch

Now we show that the implementation Π_{pke} in Figure 4 is a Δ-s-IND-PRE secure implementation of $\boldsymbol{\Sigma}_{\mathsf{pke}}$. Given any $\mathsf{Test} \in \Delta$ that is hiding w.r.t. $\boldsymbol{\Sigma}_{\mathsf{pke}}$, we need to argue that for all PPT adversary Adv,

$$\textsc{real}\langle \mathsf{Test}(0) \mid \Pi \mid \mathsf{Adv} \rangle \approx \textsc{real}\langle \mathsf{Test}(1) \mid \Pi \mid \mathsf{Adv} \rangle.$$

Our proof will need to combine the PPT indistinguishability guarantees of $\mathsf{Anon\text{-}CCA}$ of $\mathsf{pke}^\star$ along with the statistical guarantees of existential consistency, given in terms of computationally unbounded algorithms $\mathsf{pke}^\star.\mathsf{pkId}$ and $\mathsf{pke}^\star.\mathsf{msgId}$. The argument proceeds via a sequence of hybrid random variables H_i for $i = 0$ to 7, where the first hybrid is $\textsc{real}\langle \mathsf{Test}(0) \mid \Pi \mid \mathsf{Adv} \rangle$ (the output of Adv in the real world with bit $b = 0$) and the final hybrid corresponds to $\textsc{real}\langle \mathsf{Test}(1) \mid \Pi \mid \mathsf{Adv} \rangle$ (output of Adv in the real world with bit $b = 1$). Some of the hybrids refer to the extended schema $\boldsymbol{\Sigma}_{\mathsf{ext}} := (\boldsymbol{\Sigma}_{\mathsf{pke}}, \boldsymbol{\Sigma}^\dagger_{\Pi_{\mathsf{pke}}})$, where $\boldsymbol{\Sigma}^\dagger_{\Pi_{\mathsf{pke}}}$ is described below. Also, the hybrids use simulators $\mathcal{S}^\dagger_b$ (for $b \in \{0,1\}$), $\mathcal{S}^\ddagger$ and $\mathcal{S}^* \circ \mathcal{S}^\ddagger$, discussed below (with full details in Appendix I.3); of these $\mathcal{S}^*$ is a computationally unbounded simulator. We list the hybrids below.

H_0: $\textsc{real}\langle \mathsf{Test}(0) \mid \Pi_{\mathsf{pke}} \mid \mathsf{Adv} \rangle$	H_7: $\textsc{real}\langle \mathsf{Test}(1) \mid \Pi_{\mathsf{pke}} \mid \mathsf{Adv} \rangle$
H_1: $\textsc{ideal}\langle \mathsf{Test}(0) \mid \boldsymbol{\Sigma}_{\mathsf{ext}} \mid \mathcal{S}^\dagger_0 \circ \mathsf{Adv} \rangle$	H_6: $\textsc{ideal}\langle \mathsf{Test}(1) \mid \boldsymbol{\Sigma}_{\mathsf{ext}} \mid \mathcal{S}^\dagger_1 \circ \mathsf{Adv} \rangle$
H_2: $\textsc{ideal}\langle \mathsf{Test}(0) \mid \boldsymbol{\Sigma}_{\mathsf{ext}} \mid \mathcal{S}^\ddagger \circ \mathsf{Adv} \rangle$	H_5: $\textsc{ideal}\langle \mathsf{Test}(1) \mid \boldsymbol{\Sigma}_{\mathsf{ext}} \mid \mathcal{S}^\ddagger \circ \mathsf{Adv} \rangle$
H_3: $\textsc{ideal}\langle \mathsf{Test}(0) \mid \boldsymbol{\Sigma}_{\mathsf{pke}} \mid \mathcal{S}^* \circ \mathcal{S}^\ddagger \circ \mathsf{Adv} \rangle$	H_4: $\textsc{ideal}\langle \mathsf{Test}(1) \mid \boldsymbol{\Sigma}_{\mathsf{pke}} \mid \mathcal{S}^* \circ \mathcal{S}^\ddagger \circ \mathsf{Adv} \rangle$

For any $\mathsf{Test} \in \Delta$ which is s-hiding w.r.t. $\boldsymbol{\Sigma}_{\mathsf{pke}}$, our proof will establish the following:

1. First we note that $\mathsf{H}_3 \approx \mathsf{H}_4$, even though they involve a computationally unbounded simulator $\mathcal{S}^*$ (by definition of s-hiding of Test).

2. We rely on the existential consistency of the underlying PKE scheme to show that $\mathsf{H}_2 \approx \mathsf{H}_3$ and (symmetrically) $\mathsf{H}_4 \approx \mathsf{H}_5$.
3. We shall rely on the fact that $\mathsf{H}_2 \approx \mathsf{H}_5$ (established above) and the Anon-CCA security of the underlying PKE scheme to establish that $\mathsf{H}_1 \approx \mathsf{H}_2$ and (symmetrically) $\mathsf{H}_5 \approx \mathsf{H}_6$.
4. Finally, we argue that $\mathsf{H}_0 \approx \mathsf{H}_1$ and $\mathsf{H}_6 \approx \mathsf{H}_7$. This follows from the construction of $\mathcal{S}_0^\dagger$ and $\mathcal{S}_1^\dagger$ and ciphertext resistance and related properties of the PKE scheme.

Together, these steps will show that any $\mathsf{Test} \in \Delta$ that is s-hiding w.r.t. $\boldsymbol{\Sigma}_{\mathsf{pke}}$ is also hiding w.r.t. Π_{pke}. Below, we give an overview of the simulators employed above. A more detailed description of the simulators and the above steps are given in Appendix I.

Description of the Simulators. In describing the simulators $\mathcal{S}_b^\dagger, \mathcal{S}^\ddagger$ and $\mathcal{S}^*$, we use the following notation (formally defined in Appendix I). For clarity, we consider the handle-space for Test and Adv as disjoint sets $\widehat{\mathcal{H}}$ and $\overline{\mathcal{H}}$. We shall denote handles in $\widehat{\mathcal{H}}$ generically by variables like $\widehat{h}$ and $\widehat{g}$, or specifically as $\widehat{sk}$, $\widehat{pk}$ and $\widehat{ct}$, depending on the type of the handle; similarly, handles in $\overline{\mathcal{H}}$ are denoted by $\overline{h}$, $\overline{g}$, $\overline{sk}$, $\overline{pk}$ and $\overline{ct}$. Let $\mathsf{Type} : \widehat{\mathcal{H}} \cup \overline{\mathcal{H}} \to \{\mathtt{sk}, \mathtt{pk}, \mathtt{ct}\}$, denote the function mapping handles to their types. We define relations $\overset{b}{\equiv}$ and $\overset{b}{\rightsquigarrow}$ among handles in $\widehat{\mathcal{H}} \cup \overline{\mathcal{H}}$, for $b \in \{0,1\}$, as follows. For $h, g \in \widehat{\mathcal{H}} \cup \overline{\mathcal{H}}$, we say that $h \overset{b}{\equiv} g$ if, from the point of view of the ideal adversary, assuming that the test-bit given to Test equals b, h and g are supposed to correspond to the same object: i.e., if they are both obtained by transfers of the same handle, or are both public-key handles derived from secret-key handles that are derived from the same handle via transfers and clonings. We define $h \overset{b}{\rightsquigarrow} g$ to indicate that the handle g was *derived from an original (secret-key) handle* h, by operations including transfers, public-key derivations (possibly after clonings) and encryptions.

$\mathcal{S}_b^\dagger$ is a simple and efficient simulator which knows the test-bit b (and hence knows the exact agents created by Test), which has two jobs: firstly, it generates simulated secret-keys on behalf of Test and uses them to simulate the objects transferred by Test; secondly, it recognizes any object that the user transfers to Test as being derived from such a secret-key. For adversarial objects which are not derived from the simulated secret-keys, $\mathcal{S}_b^\dagger$ will run the accept algorithm (required by existential consistency) and simply forward those objects that are accepted directly to the extension $\boldsymbol{\Sigma}^\dagger_{\Pi_{\mathsf{pke}}}$. We heavily rely on existential consistency properties to argue that if $\mathcal{S}_b^\dagger$ classifies an object (e.g., a public key) transferred from the adversary as not related to any simulated secret-key, then this object and any objects derived from it by Test will not end up "matching" any simulated secret-key later on.

$\mathcal{S}^\ddagger$ is also an efficient simulator, but it does not know the test-bit b. It will rely on the hiding guarantee of Test to carry out a simulation without using b. $\mathcal{S}^\ddagger$ uses a lazy strategy to assign objects to handles transferred by Test. For instance, if Test reports that it is carrying out $(\mathsf{transfer}, \widehat{h}_0, \widehat{h}_1)$ (i.e., transferring the agent corresponding to the handle $\widehat{h}_b$), at that point $\mathcal{S}^\ddagger$ tries to come up with an object obj that it transfers to the user. This means that if $b = 0$, the handle $\widehat{h}_0$ is assigned obj and if $b = 1$ a (possibly different handle) $\widehat{h}_1$ is assigned the same object. The relations $\overset{0}{\equiv}$, $\overset{1}{\equiv}$, $\overset{0}{\rightsquigarrow}$ and $\overset{1}{\rightsquigarrow}$ are used to maintain a valid view for the user. $\mathcal{S}^\ddagger$ is designed so that when it gets stuck it corresponds to a conflict between the $b = 0$ and $b = 1$ scenarios, which will reveal b. Since Test is a hiding test, $\mathcal{S}^\ddagger$ will get stuck only with negligible probability. When it does not get stuck it is able to carry out a good simulation by relying on Anon-CCA-security of $\mathsf{pke}^\star$.

Finally, $\mathcal{S}^*$ is a computationally unbounded simulator. Intuitively, its job is to remove the need for the extension $\boldsymbol{\Sigma}^\dagger_{\Pi_{\mathsf{pke}}}$. It forces open the objects that were being sent to $\boldsymbol{\Sigma}^\dagger_{\Pi_{\mathsf{pke}}}$ using (inefficient) algorithms pkId and msgId guaranteed by existential consistency; this lets it translate these objects into ideal agents and commands that can be sent to $\boldsymbol{\Sigma}_{\mathsf{pke}}$. Existential consistency conditions imply that the execution using $\boldsymbol{\Sigma}^\dagger_{\Pi_{\mathsf{pke}}}$ (on objects that were accepted by accept) will remain faithful to the execution in $\boldsymbol{\Sigma}_{\mathsf{pke}}$.

8 Conclusion and Future Work

Our concrete technical contribution in this paper relates to public-key encryption, and gives a significantly stronger definition which can yet be achieved with very modest overheads.

Along the way, we significantly extended the Cryptographic Agents framework [2] to allow modeling adversarially created objects. Borrowing the IND-PRE security definition from this framework allowed us to push the limits of the real/ideal paradigm by going beyond the simulation-based security definition.

Admittedly, in this work we have not fully explored the possibilities of such a new framework, as our focus was on a single (albeit important) primitive. We leave it for future work to investigate the power and limitations of this framework, and enhance it as appropriate. In particular, signatures could also be modeled in this framework and one could hope that, as in the case of encryption, a simpler equivalent definition can be formulated to avoid the need for tedious proofs for individual constructions. We also leave it for future work to enhance the framework to allow composition across multiple schemas (e.g., signing of encrypted messages, or even encrypting ciphertexts themselves). In our current model, issues like circular security do not fully arise (though it does permit the adversary to prepare encryption cycles), but a model allowing arbitrary composition will need to address such issues.

This framework also provides an elegant approach to defining security for new and emerging cryptographic primitives like obfuscation and functional encryption. Indeed, this was the original motivation in [2], though in their model only CPA security was captured. Given that our model supports CCA security (and more) without succumbing to the impossibility results for simulation-based security, we have a chance to reexamine the definitions of these new primitives and possibly strengthen them.

Another direction for future research is to leverage the ideal model in the Cryptographic Agents framework to simplify *automated analysis* of complex security systems involving various cryptographic objects. This is an emerging area of much practical interest, and we believe that the extended Cryptographic Agents framework can provide guidance for practical design that supports rigorous analysis.

References

1. Michel Abdalla, Mihir Bellare, and Gregory Neven. Robust encryption. In Daniele Micciancio, editor, *TCC*, 2010.
2. Shashank Agrawal, Shweta Agrawal, and Manoj Prabhakaran. Cryptographic agents: Towards a unified theory of computing on encrypted data. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT*, 2015.
3. Shashank Agrawal, Manoj Prabhakaran, and Ching-Hua Yu. Virtual grey-boxes beyond obfuscation: A statistical security notion for cryptographic agents. In *TCC 2016-B*, 2016.
4. Mihir Bellare, Alexandra Boldyreva, Anand Desai, and David Pointcheval. Key-privacy in public-key encryption. In *Asiacrypt*, pages 566–582. Springer, 2001.
5. Mihir Bellare, David Cash, and Rachel Miller. Cryptography secure against related-key attacks and tampering. In Dong Hoon Lee and Xiaoyun Wang, editors, *Advances in Cryptology – ASIACRYPT*, 2011.
6. Mihir Bellare, Dennis Hofheinz, and Eike Kiltz. Subtleties in the definition of IND-CCA: when and how should challenge decryption be disallowed? *J. Cryptology*, 28(1):29–48, 2015.
7. Daniel Bleichenbacher. Chosen ciphertext attacks against protocols based on the RSA encryption standard PKCS #1. 1462, 1998.
8. Manuel Blum, Paul Feldman, and Silvio Micali. Non-interactive zero-knowledge and its applications (extended abstract). In *STOC*, pages 103–112, 1988.
9. R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *Proceedings of the 42nd IEEE Symposium on Foundations of Computer Science*, FOCS '01, 2001.
10. Ran Canetti, Uriel Feige, Oded Goldreich, and Moni Naor. Adaptively secure multi-party computation. In *STOC*, pages 639–648, 1996.
11. Ran Canetti and Jonathan Herzog. Universally composable symbolic analysis of mutual authentication and key-exchange protocols. In *TCC*, pages 380–403, 2006.
12. Ran Canetti, Hugo Krawczyk, and Jesper Buus Nielsen. Relaxing chosen-ciphertext security. In *CRYPTO*, pages 565–582, 2003.
13. Sandro Coretti, Ueli Maurer, and Björn Tackmann. Constructing confidential channels from authenticated channels–public-key encryption revisited. In *ASIACRYPT*, pages 134–153, 2013.
14. Ronald Cramer and Victor Shoup. A practical public key cryptosystem provably secure against adaptive chosen ciphertext attack. In *CRYPTO*, volume 1462 of *Lecture Notes in Computer Science*. Springer, 1998.
15. Danny Dolev, Cynthia Dwork, and Moni Naor. Nonmalleable cryptography. *SIAM Journal on Computing*, 30(2):391–437 (electronic), 2000. Preliminary version in STOC 1991.

16. Cynthia Dwork, Moni Naor, and Amit Sahai. Concurrent zero-knowledge. *J. ACM*, 51(6):851–898, 2004. Preliminary version in STOC'98.
17. Pooya Farshim, Benoît Libert, Kenneth G. Paterson, and Elizabeth A. Quaglia. Robust encryption, revisited. In Kaoru Kurosawa and Goichiro Hanaoka, editors, *PKC*, 2013.
18. Oded Goldreich. *Foundations of Cryptography: Volume 1*. Cambridge University Press, New York, NY, USA, 2006.
19. Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play ANY mental game. In *STOC*, 1987.
20. Shafi Goldwasser and Silvio Micali. Probabilistic encryption. *Journal of Computer and System Sciences*, 28(2):270–299, April 1984. Preliminary version appeared in STOC' 82.
21. Shai Halevi. A sufficient condition for key-privacy. *IACR Cryptology ePrint Archive*, 2005:5, 2005.
22. Ryotaro Hayashi and Keisuke Tanaka. Universally anonymizable public-key encryption. In *Advances in Cryptology - ASIACRYPT 2005*, pages 293–312, 2005.
23. Ryotaro Hayashi and Keisuke Tanaka. Schemes for encryption with anonymity and ring signature. *IEICE Transactions*, 89-A(1):66–73, 2006.
24. M. Joye. A key-private cryptosystem from the quadratic residuosity. In *2015 12th International Joint Conference on e-Business and Telecommunications (ICETE)*, volume 04, pages 398–404, July 2015.
25. Philip Mackenzie, Michael K. Reiter, and Ke Yang. Alternatives to Non-malleability: Definitions, Constructions, and Applications. In *TCC*, pages 171–190, 2004.
26. Ueli Maurer. Constructive cryptography - A new paradigm for security definitions and proofs. In *Theory of Security and Applications - Joint Workshop, TOSCA 2011*, pages 33–56, 2011.
27. Payman Mohassel. A closer look at anonymity and robustness in encryption schemes. In *ASIACRYPT*, pages 501–518, 2010.
28. Moni Naor and Moti Yung. Public-key cryptosystems provably secure against chosen ciphertext attacks. In *STOC*, pages 427–437, 1990.
29. Roger M. Needham and Michael D. Schroeder. Using encryption for authentication in large networks of computers. *Commun. ACM*, 21(12):993–999, 1978.
30. Akshay Patil. On symbolic analysis of cryptographic protocols. Master's thesis, Massachusetts Institute of Technology, 2005.
31. Manoj Prabhakaran and Mike Rosulek. Rerandomizable rcca encryption. In Alfred Menezes, editor, *Advances in Cryptology - CRYPTO*, 2007.
32. Brent R. Waters, Edward W. Felten, and Amit Sahai. Receiver anonymity via incomparable public keys. In *ACM Conference on Computer and Communications Security, CCS*, pages 112–121, New York, NY, USA, 2003. ACM.
33. Hoeteck Wee. Public key encryption against related key attacks. In Marc Fischlin, Johannes Buchmann, and Mark Manulis, editors, *Public Key Cryptography – PKC*, 2012.

A Additional Notes on Related Work

Public key encryption is one of the most fundamental primitives of cryptography, and its security has been investigated thoroughly. Aside from the classic CPA/CCA security notions, security for public key encryption has been extended in many ways. One important example is the notion of *anonymity* (also known as *key privacy*) introduced by Bellare, Boldyreva, Desai and Pointcheval [4], and studied in several subsequent works [21,22,23,24].

Robustness was recently introduced to address security concerns related to decrypting ciphertexts with "wrong" secret-keys [1,27]. These works also discussed motivating applications, including a previously proposed auction protocol, where these concerns become relevant. Motivated by some of the same applications, Farshim et al. further strengthened the notion to also consider the case of adversarially generated secret-keys [17]. Full Robustness introduced in [17] is a special case of COA security. Farshim et al. [17] introduced a stronger notion called Complete Robustness as well, which is concerned additionally with using a PKE scheme as a commitment scheme, wherein opening is carried out by revealing the randomness used for encryption. (As mentioned above, by design, COA does not deal with such uses of a PKE.)

There have also been works that model PKE in general frameworks like UC security [11,30] and Constructive Cryptography [13], but they typically consider only the case of honestly generated keys (with honest parties' secret-keys never leaving them). In fact, as these works use strong simulation-based definitions, it is impossible to achieve their notion of security if the secret-keys can be transferred.

Another related problem (but not addressed in this work) is that of *related key attacks* [5,33]. In this setting, an adversary may be able to inject faults into the secret key and obtain decryptions of ciphertexts under this modified secret key. While COA security does not deal with tampering or leakage, the two problems are related in the sense that they both consider honest parties carrying out decryption under adversarially created/manipulated secret-keys.

There have been several other works extending the notion of security for PKE which we have not discussed. We also note that weaker notions than CCA security for public key encryption that suffice for various applications have also been considered (e.g. [12,25,31]) but our focus is on stronger security.

The cryptographic agents framework was proposed in [2], and used in [3], to reason about advanced primitives like obfuscation and functional encryption. We significantly revise and extend the framework to be able to handle adversarially generated objects.

B Additional Example of an Attack

The following (somewhat elaborate) scenario illustrates the threat if an adversary may generate multiple seemingly unrelated public-keys that all produce ciphertexts that correctly decrypt under the secret-key of a given public-key.

Suppose in a distributed cloud storage system, Alice uploads encrypted secret-shares of a document, with each share encrypted under a different public-key, and the (anonymous) owner of each public-key will locally decrypt and store every share it can decrypt. Now, suppose an adversary who partly compromises the system can change the set of public-keys used by Alice to encrypt the shares, but does not get access to the uploaded ciphertexts themselves. Instead, it has gained access to the shares decrypted by one of the decrypting parties, Bob. With just Alice and Bob compromised in this limited manner, we could still expect the adversary not to have access to Alice's document, as only one of the shares will be encrypted under Bob's public-key.

However, the following attack allows the adversary to obtain Alice's document: The adversary derives several distinct public-keys that are all equivalent to Bob's public-key, and replaces the public-keys in Alice's hand with them. Since they look different, Alice unsuspectingly encrypts her shares under these public-keys. Bob retrieves these ciphertexts, and all of them successfully decrypt, leading him to place them in his local store, from which the adversary can recover Alice's document.

C Formalism of Agents

For the sake of completeness, we include a formalism for modeling agents and sessions, borrowed from [2] (with minor changes).

Definition 6 (Agents). *An agent is an interactive Turing Machine, with the following modifications:*

- *There is an a priori restriction on the size of all the tapes other than the randomness tape (including input, communication and work tapes), as a function of the security parameter.*
- *There is a special* blocking state *such that if the machine enters such a state, it remains there if the input tape is empty. Similarly, there are blocking states which let the machine block if any combination of the communication tape and the input tape is empty.*

We can allow *non-uniform agents* by allowing an additional advice tape. Our framework and basic results work in the uniform and non-uniform model equally well.

Note that an agent who enters a blocking state can move out of it if its configuration is changed by adding a message to its input tape and/or communication tape. However, if the agent enters a halting state, it will not move out of that state. An agent who never enters a blocking state is called a *non-reactive agent*. An agent who never reads or writes from a communication tape is called a *non-interactive agent*.

Definition 7 (Session). *A session maps a finite ordered set of agents, their configurations and inputs, to outputs and (updated) configurations of the same agents, as follows. The agents are initialized with the given inputs on their input tapes, and then executed together until they are deadlocked.*[19] *The result of applying the session is defined as the collection of outputs and configurations of the agents when the session terminates (if it terminates; if not, the result is left undefined).*

We shall be restricting ourselves to collections of agents such that sessions involving them are guaranteed to terminate. Note that we have defined a session to have only an initial set of inputs, so that the outcome of a session is well-defined (without the need to specify how further inputs would be chosen).

D Cryptographic Primitives Used

D.1 Public Key Encryption

We recall the definition of public key encryption.

Below, for convenience we shall separate the key generation into two parts – a secret-key generation algorithm $\mathsf{pke.skGen}$ and a deterministic public-key derivation algorithm $\mathsf{pke.pkGen}$. Also, we shall allow for an infinite message space, but will specify a "length" function length of the messages that can be leaked by the ciphertexts.

Definition 8. *A public-key encryption scheme* pke *is specified by sets* $\mathcal{SK}$, $\mathcal{PK}$, $\mathcal{CT}$ *and* $\mathcal{M}$ *(the space of secret keys, public keys, ciphertexts and messages, respectively), and four polynomial time algorithms* $\mathsf{pke.skGen}$, $\mathsf{pke.pkGen}$, $\mathsf{pke.encrypt}$ *and* $\mathsf{pke.decrypt}$ *that satisfy the conditions given below. (All the algorithms receive the security parameter as an additional input.)*

- $\mathsf{pke.skGen}$ *is a randomized algorithm that outputs a secret key* $\mathsf{SK} \in \mathcal{SK}$.
- $\mathsf{pke.pkGen} : \mathsf{SK} \mapsto \mathsf{PK}$. $\mathsf{pke.pkGen}$ *is a deterministic algorithm that on input a secret key* $\mathsf{SK} \in \mathcal{SK}$*, outputs a public key* $\mathsf{PK} \in \mathcal{PK}$.
- $\mathsf{pke.encrypt} : (\mathsf{PK}, m) \to \mathsf{CT}$. $\mathsf{pke.encrypt}$ *is a randomized algorithm that on input* $\mathsf{PK} \in \mathcal{PK}$ *and a message* $m \in \mathcal{M}$*, outputs a ciphertext* $\mathsf{CT} \in \mathcal{CT}$.
- $\mathsf{pke.decrypt}(\mathsf{SK}, \mathsf{CT}) \to m$. $\mathsf{pke.decrypt}$ *is a deterministic algorithm that on input the secret key* $\mathsf{SK} \in \mathcal{SK}$ *and a ciphertext* $\mathsf{CT} \in \mathcal{CT}$*, outputs* $m \in \mathcal{M} \cup \{\perp\}$.

[19] More precisely, the first agent is executed till it enters a blocking or halting state, and then the second and so forth, in a round-robin fashion, until all the agents remain in blocking or halting states for a full round. After each execution of an agent, the contents of its outgoing communication tape are interpreted as an ordered sequence of messages to each of the other agents in the session (some or all of them possibly being empty messages), and copied over to the respective agents' incoming communication tapes.

Correctness. For all messages $m \in \mathcal{M}$, the probability that when

$$\mathsf{SK} \leftarrow \mathsf{pke.skGen}, \quad \mathsf{pke.decrypt}(\mathsf{SK}, \mathsf{pke.encrypt}(\mathsf{pke.pkGen}(\mathsf{SK}), m)) \neq m$$

is negligible in κ, where the probability is taken over the randomness used in the pke.skGen *and* pke.encrypt *algorithms.*

For brevity, we shall use pke.keyGen to be the following combination of pke.skGen and pke.pkGen: it samples $\mathsf{SK} \leftarrow \mathsf{pke.skGen}$ and outputs $(\mathsf{SK}, \mathsf{pke.pkGen}(\mathsf{SK}))$. [20]

In order to define security for an infinite message-space, we assume that there is an efficiently computable function $\mathsf{length} : \mathcal{M} \to \mathbb{Z}_+$ which can be revealed by a ciphertext.

Definition 9 (Anonymous CCA Security). *A public-key encryption scheme* pke *is anonymous chosen-plaintext attack (*Anon-CCA*) secure if for all* PPT *adversaries $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$, probability that the experiment $\mathsf{Expt}^{\mathsf{ano\text{-}cca}}_{\mathcal{A}}(\kappa)$ (defined in Figure 5) outputs 1 is at most $\frac{1}{2} + \epsilon(\kappa)$ where ϵ is negligible in κ.*

$\mathsf{Expt}^{\mathsf{ano\text{-}cca}}_{\mathcal{A}}(\kappa)$:

1. $(\mathsf{SK}_0, \mathsf{PK}_0) \leftarrow \mathsf{pke.keyGen}$; $(\mathsf{SK}_1, \mathsf{PK}_1) \leftarrow \mathsf{pke.keyGen}$
2. $(m_0, m_1, \mathsf{state}) \leftarrow \mathcal{A}_1^{\mathsf{pke.decrypt}(\mathsf{SK}_0,\cdot),\mathsf{pke.decrypt}(\mathsf{SK}_1,\cdot)}(1^\kappa, \mathsf{PK}_0, \mathsf{PK}_1)$
3. $b \leftarrow_R \{0,1\}$ and $\mathsf{CT}^* \leftarrow \mathsf{pke.encrypt}(\mathsf{PK}_b, m_b)$
4. $b' \leftarrow \mathcal{A}_2^{\mathsf{pke.decrypt}(\mathsf{SK}_0,\cdot),\mathsf{pke.decrypt}(\mathsf{SK}_1,\cdot)}(\mathsf{state}, \mathsf{CT}^*)$. Now, both oracles return $\perp$ if queried with CT^*.
5. Output $b \oplus b'$, if $m_0, m_1 \in \mathcal{M}$, $\mathsf{length}(m_0) = \mathsf{length}(m_1)$ and $b' \in \{0,1\}$. Else, output a random bit.

Fig. 5 Key-anonymous chosen-ciphertext attack security for PKE

D.2 Augmented Key Anonymous CCA Security

It will be convenient for us to make use of the following augmented anonymous CCA game. Security with respect to this game is implied by the original Anon-CCA security formulation in Figure 5.

A public-key encryption scheme pke is an augmented anonymous chosen-plaintext attack (Anon-CCA) secure if for all PPT adversaries $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$, probability that the experiment $\mathsf{Expt}^{\mathsf{ano\text{-}cca}}_{\mathcal{A}}(\kappa)$ defined in Figure 6 outputs 1 is at most $\frac{1}{2} + \epsilon(\kappa)$ where ϵ is negligible in κ.

D.3 Commitment schemes

Definition 10. *A (non-interactive) commitment scheme for a message space $\mathcal{M} = \{\mathcal{M}_\kappa\}_{\kappa \in \mathbb{N}}$ consists of two polynomial time algorithms defined below:*

- $\mathsf{com.Commit}(1^\kappa, m) \to (\mathsf{c}, \mathsf{d})$. com.Commit *is a randomized algorithm that on input a message $m \in \mathcal{M}_\kappa$, outputs a commitment* c *and decommitment information* d*.*
- $\mathsf{com.Open}(1^\kappa, \mathsf{c}, \mathsf{d}) \to m$ *or* $\perp$. com.Open *is a deterministic algorithm that on input a commitment* c *and decommitment information* d*, outputs a message $m \in \mathcal{M}_\kappa$ or $\perp$.*

[20] The conventional presentation of a PKE scheme is in terms of pke.keyGen which outputs a pair of keys $(\mathsf{SK}, \mathsf{PK})$. Such a scheme can be made to fit the above definition by defining pke.skGen to be identical to pke.keyGen (treating $\mathsf{SK}' := (\mathsf{SK}, \mathsf{PK})$ as the secret-key) and defining pke.pkGen as the algorithm which simply takes $\mathsf{SK}' = (\mathsf{SK}, \mathsf{PK})$ and outputs $\mathsf{PK}' := \mathsf{PK}$.

$\mathsf{Expt}_{\mathcal{A}}^{\mathsf{ano\text{-}cca}}(\kappa)$:

1. For $i \in \{0\} \cup [k]$, $(\mathit{SK}_i, \mathit{PK}_i) \leftarrow \mathsf{pke.keyGen}$.
2. $(\mathit{index}, m_0, m_1, \mathit{state}) \leftarrow \mathcal{A}_1^{\mathcal{O}, \{\mathsf{pke.decrypt}(\mathit{SK}_i, \cdot)\}_{i \in [0,k]}}(\mathit{PK}_0, \ldots, \mathit{PK}_k)$. Here, $\mathit{index} \in [k]$ and $\mathcal{O}$ is an oracle which when queried on $j \in [k]$ and returns SK_j.
3. $b \leftarrow_R \{0,1\}$. If $b = 0$, let $\mathit{CT}^* \leftarrow \mathsf{pke.encrypt}(\mathit{PK}_0, m_0)$. If $b = 1$, let $\mathit{CT}^* \leftarrow \mathsf{pke.encrypt}(\mathit{PK}_{\mathsf{index}}, m_1)$
4. $b' \leftarrow \mathcal{A}_2^{\{\mathsf{pke.decrypt}(\mathit{SK}_i, \cdot)\}_{i \in [0,k]}, \mathcal{O}}(\mathit{state}, \mathit{CT}^*, \mathit{PK}_0, \ldots, \mathit{PK}_k)$. Now all the decryption oracles return $\perp$ if queried with CT^* and $\mathcal{O}$ returns $\perp$ on query index.
5. Output $b \oplus b'$, if $m_0, m_1 \in \mathcal{M}$, $\mathsf{length}(m_0) = \mathsf{length}(m_1)$ and $b' \in \{0,1\}$. Else, output a random bit.

Fig. 6 Augmented experiment for Key-Anonymous Chosen-Ciphertext Attack Security

Correctness. For every $m \in \mathcal{M}_\kappa$, $\mathsf{com.Open}(\mathsf{com.Commit}(m)) = m$, irrespective of the randomness used in $\mathsf{com.Commit}$.

For security, we require the following properties:

Perfectly binding. A commitment scheme is perfectly binding if for any two strings $c^\star, d^\star \in \{0,1\}^\star$, there do not exist $m_0, m_1 \in \mathcal{M}_\kappa$ and coin tosses r_0, r_1 such that $m_0 \neq m_1$ and $(c^\star, d^\star) = \mathsf{com.Commit}(m_0; r_0) = \mathsf{com.Commit}(m_1; r_1)$.

Computational hiding. A commitment scheme is computationally hiding if for all $m_0, m_1 \in \mathcal{M}_\kappa$ and all PPT adversaries $\mathcal{A}$,

$$|\Pr[\mathcal{A}(1^\kappa, \mathsf{com.Commit}(m_0)) = 1] - \Pr[\mathcal{A}(1^\kappa, \mathsf{com.Commit}(m_1)) = 1]| \leq \mathrm{negl}(\kappa).$$

For one-bit messages, such commitment schemes can be constructed based on any 1-1 one-way function (Construction 4.4.2 in [18]) . In order to commit to a string, one can commit to each bit individually (using hard-core functions could give better efficiency).

D.4 Signature schemes

Definition 11. *A signature scheme for a message space $\mathcal{M} = \{\mathcal{M}_\kappa\}_{\kappa \in \mathbb{N}}$ is a triple of polynomial time algorithms that satisfy a correctness condition given below. (All the algorithms receive the security parameter as input.)*

- $\mathsf{sig.keyGen} \to (\mathit{sigk}, \mathit{verk})$. $\mathsf{sig.keyGen}$ *is a randomized algorithm that on input the security parameter, outputs a signing key* sigk *and a verification key* verk.
- $\mathsf{Sign}(\mathit{sigk}, m) \to \tau$. Sign *is a randomized algorithm that takes the signing key* sigk *and a message $m \in \mathcal{M}_\kappa$ as inputs, and outputs a signature τ.*
- $\mathsf{Verify}(\mathit{verk}, m, \tau) \to \{0,1\}$. Verify *is a deterministic algorithm that on input the verification key* verk*, a message, and a signature τ, outputs a binary value (where 1 denotes success and 0 failure).*

Correctness. For all messages $m \in \mathcal{M}_\kappa$, the probability that when

$$(\mathit{sigk}, \mathit{verk}) \leftarrow \mathsf{sig.keyGen}, \quad \mathsf{Verify}(\mathit{verk}, m, \mathsf{Sign}(\mathit{sigk}, m)) = 0$$

is negligible in κ, where the probability is taken over the randomness used in the $\mathsf{sig.keyGen}$ *and* Sign *algorithms.*

Strong Existential Unforgeability for One-Time Signatures. A one-time signature scheme is *strongly existentially unforgeable* if for all PPT adversaries $\mathcal{A}$, probability that the experiment $\mathsf{Expt}_{\mathcal{A}}^{\mathsf{sig}}(\kappa)$ defined in Figure 7 outputs 1 is negligible in κ.

$\mathsf{Expt}^{\mathsf{sig}}_{\mathcal{A}}(\kappa)$:

- $(\mathit{sigk}, \mathit{verk}) \leftarrow \mathsf{sig.keyGen}$
- $(m_0, \mathit{state}) \leftarrow \mathcal{A}(\mathit{verk})$
- $\tau_0 \leftarrow \mathsf{sig.Sign}(\mathit{sigk}, m_0)$
- $(m, \tau) \leftarrow \mathcal{A}(\mathit{state}, \tau_0)$
- If $(m, \tau) \neq (m_0, \tau_0)$ and $\mathsf{sig.Verify}(\mathit{verk}, m, \tau) = 1$, output 1 else 0.

Fig. 7 Strong Existential Unforgeability of One-Time Signatures

Given a PKE scheme, $\mathsf{pke} = (\mathsf{pke.skGen}, \mathsf{pke.pkGen}, \mathsf{pke.encrypt}, \mathsf{pke.decrypt})$ constructing a PKE scheme $\mathsf{pke}^\star$. We use the same conventions as in Figure 1 for the algorithms below.

- $\mathsf{pke}^\star.\mathsf{skGen}$. It outputs $(r_{\mathsf{pke}}, r_{\mathsf{com}})$, where r_{pke} and r_{com} are as in Figure 1.
- $\mathsf{pke}^\star.\mathsf{pkGen}(\mathit{SK}^\star)$. This is a deterministic function. On input $\mathit{SK}^\star$, parse it as $(r_{\mathsf{pke}}, r_{\mathsf{com}})$. Let $\mathit{SK} \leftarrow \mathsf{pke.skGen}$ using random-tape r_{pke}. Compute $\mathit{PK} \leftarrow \mathsf{pke.pkGen}(\mathit{SK})$ and $(\mathit{c}, \mathit{d}) := \mathsf{com.Commit}(\mathit{SK}; r_{\mathsf{com}})$. Output $(\mathit{PK}, \mathit{c})$ (or $\perp_{\text{PK}}$ if any of the steps fails).
- $\mathsf{pke}^\star.\mathsf{encrypt}(\mathit{PK}^\star, m)$. Let $(\mathit{c}^\star, \mathit{d}^\star) \leftarrow \mathsf{com.Commit}(\mathit{PK}^\star)$. Let $\mu = (\mathit{c}^\star, \mathit{d}^\star, m)$. Parse $\mathit{PK}^\star$ as $(\mathit{PK}, \mathit{c})$, and output $(\mathit{c}^\star, \mathsf{pke.encrypt}(\mathit{PK}, \mu))$ as the ciphertext.
- $\mathsf{pke}^\star.\mathsf{decrypt}(\mathit{SK}^\star, \mathit{CT}^\star)$. Parse $\mathit{CT}^\star$ as $(\mathit{c}^\star, \mathit{CT})$ and $\mathit{SK}^\star$ as $(r_{\mathsf{pke}}, r_{\mathsf{com}})$. Then do the following:
  1. Compute $\mathit{PK}^\star := \mathsf{pke}^\star.\mathsf{pkGen}(\mathit{SK}^\star)$. Along the way, this obtains SK by running $\mathsf{pke.skGen}$ with random-tape r_{pke}.
  2. Decrypt CT with SK to obtain $\mu = (\widetilde{c}, \widetilde{d}, m)$.
  3. Check if $\widetilde{c} = \mathit{c}^\star$ and $\mathit{PK}^\star = \mathsf{com.Open}(\widetilde{c}, \widetilde{d})$. If all the steps succeed, output m (else output $\perp$).
- $\mathsf{pke}^\star.\mathsf{accept}(\mathit{obj})$. It merely checks if obj has the form of a secret-key, public-key or ciphertext, in which case it outputs SK, PK or CT respectively. Otherwise it outputs $\perp$.

Fig. 8 A COA secure PKE scheme.

E Proof Omitted from Section 4.1

In this section we argue that the PKE scheme in Figure 8 satisfies COA security. Recall that for this we need to show that the scheme satisfies existential consistency (Definition 1) and Anon-CCA (Definition 9).

Existential consistency. Firstly, we note that the first set of conditions in Definition 1 is satisfied because $\mathsf{pke}^\star.\mathsf{accept}$ (which is defined to check only the format) accepts any output from $\mathsf{pke}^\star.\mathsf{skGen}$, $\mathsf{pke}^\star.\mathsf{pkGen}$ and $\mathsf{pke}^\star.\mathsf{encrypt}$ as SK, PK and CT respectively, with probability 1. To see that the second condition is also satisfied, we define $\mathsf{pke}^\star.\mathsf{pkId}$ and $\mathsf{pke}^\star.\mathsf{msgId}$ as follows: $\mathsf{pke}^\star.\mathsf{pkId}(\mathit{c}, \mathit{CT}^\star)$ simply retrieves the message $\mathit{PK}^\star$ in the perfectly binding commitment c (or outputs $\perp_{\text{PK}}$ if no such message exists). $\mathsf{pke}^\star.\mathsf{msgId}(\mathit{CT}^\star)$ further identifies an arbitrary secret-key $\mathit{SK}^\star$ such that $\mathsf{pke}^\star.\mathsf{pkGen}(\mathit{SK}^\star) = \mathit{PK}^\star$ and outputs $\mathsf{pke}^\star.\mathsf{decrypt}(\mathit{SK}^\star, \mathit{CT}^\star)$ (or $\perp$ if no such $\mathit{SK}^\star$ exists). Below, we consider the three probabilities in the second item and show that they are all negligible. The two do not rely on universal key reliability, while the third does.

Consider any $\mathit{SK}^\star$ and $\mathit{CT}^\star$ such that there exists $\mathit{PK}^\star_0 \neq \perp_{\text{PK}}$, with $\mathsf{pke}^\star.\mathsf{pkGen}(\mathit{SK}^\star) = \mathsf{pke}^\star.\mathsf{pkId}(\mathit{CT}^\star) = \mathit{PK}^\star_0$. Further, suppose $\mathsf{pke}^\star.\mathsf{msgId}(\mathit{CT}^\star)$ identifies a key $\mathit{SK}^\star_0$ such that $\mathsf{pke}^\star.\mathsf{skGen}(\mathit{SK}^\star_0) = \mathit{PK}^\star_0$ (such a key exists since $\mathsf{pke}^\star.\mathsf{skGen}(\mathit{SK}^\star) = \mathit{PK}^\star_0$). Then $D(\mathit{SK}^\star, \mathit{CT}^\star) = \mathsf{pke}^\star.\mathsf{decrypt}(\mathit{SK}^\star_0, \mathit{CT}^\star)$. We need to ensure that this equals $\mathsf{pke}^\star.\mathsf{decrypt}(\mathit{SK}^\star, \mathit{CT}^\star)$. Since $\mathsf{pke}^\star.\mathsf{skGen}(\mathit{SK}^\star_0) = \mathsf{pke}^\star.\mathsf{skGen}(\mathit{SK}^\star)$, we can write $\mathit{SK}^\star_0 = (r^0_{\mathsf{pke}}, r^0_{\mathit{c}})$ and $\mathit{SK}^\star = (r^0_{\mathsf{pke}}, r^0_{\mathit{c}})$, where $r_{\mathsf{pke}} = r^0_{\mathsf{pke}}$. Then, from the description of $\mathsf{pke}^\star.\mathsf{decrypt}$ in Figure 8, it can be seen that $\mathsf{pke}^\star.\mathsf{decrypt}$ computes the same intermediate values $(\mathit{PK}^\star, \mathit{SK}, \widetilde{c}, \widetilde{d}, m)$ when decrypting a ciphertext using $\mathit{SK}^\star_0$ or $\mathit{SK}^\star$. This ensures that if $\mathsf{pke}^\star.\mathsf{pkGen}(\mathit{SK}^\star) = \mathsf{pke}^\star.\mathsf{pkId}(\mathit{CT}^\star) \neq \perp$, then $\mathsf{pke}^\star.\mathsf{decrypt}(\mathit{SK}^\star, \mathit{CT}^\star) = D(\mathit{SK}^\star, \mathit{CT}^\star)$. Further, if $\mathsf{pke}^\star.\mathsf{pkGen}(\mathit{SK}^\star) \neq \mathsf{pke}^\star.\mathsf{pkId}(\mathit{CT}^\star)$ (or if they are both $\perp_{\text{PK}}$), then $\mathsf{pke}^\star.\mathsf{decrypt}(\mathit{SK}^\star, \mathit{CT}^\star) = \perp$, since $\mathsf{pke}^\star.\mathsf{decrypt}$ ensures that the statistically binding commitment in $\mathit{CT}^\star$ (which equals $\mathsf{pke}^\star.\mathsf{pkId}(\mathit{CT}^\star)$) should equal $\mathsf{pke}^\star.\mathsf{pkGen}(\mathit{SK}^\star)$ before it outputs a message other than $\perp$.

The second condition follows directly from the description of the encryption algorithm (which includes a commitment to $\mathsf{PK}^\star$ in the ciphertext) and the algorith $\mathsf{pke}^\star.\mathsf{pkId}$ (which extracts the value thus committed).

The third property relies on universal key reliability. Note that if $\mathsf{pke}^\star.\mathsf{accept}(\mathsf{SK}^\star) = \textsc{sk}$, then $\mathsf{SK}^\star = (r_{\mathsf{pke}}, r_{\mathsf{c}})$ of the appropriate lengths. Universal key reliability of the underlying scheme pke for $\mathsf{SK} = \mathsf{pke.skGen}$ using any random string r_{pke} and $\mathsf{PK} = \mathsf{pke.pkGen}(\mathsf{SK})$, for all messages m, $\mathsf{pke.decrypt}(\mathsf{pke.encrypt}(\mathsf{PK}, m)) = m$ except with negligible probability. The rest of the decryption algorithm $\mathsf{pke}^\star.\mathsf{decrypt}$ is guaranteed to proceed without errors, since the commitment scheme is perfectly correct.

$\mathsf{Anon\text{-}CCA}$ *security.* To prove $\mathsf{Anon\text{-}CCA}$ security of $\mathsf{pke}^\star$, first we prove that it is CCA secure. For this we shall reduce CCA security of pke to that of $\mathsf{pke}^\star$, as follows. Firstly, a public-key in pke needs to be translated to a public-key in $\mathsf{pke}^\star$. This cannot be done perfectly, since a public-key in $\mathsf{pke}^\star$ is of the form $\mathsf{PK}^\star = (\mathsf{PK}, \mathsf{c})$ where c is a commitment to the underlying secret-key SK! However, we can modify $\mathsf{pke}^\star.\mathsf{decrypt}$ to not use $\mathsf{SK}^\star$ as it is, but only use c and oracle access to $\mathsf{pke.decrypt}(\mathsf{SK}, \cdot)$. Then, thanks to the hiding of commitment, we can replace c to be the commitment of a dummy message and the entire CCA security experiment remains indistinguishable. It is an adversary in this modified experiment that we use to attack the CCA security of pke. In the modified game, we can indeed take a public-key PK for pke and produce a public-key $\mathsf{PK}^\star = (\mathsf{PK}, \mathsf{c})$, taking c to be the commitment to a dummy message. A message pair given by the adversary for encryption in the CCA security experiment for $\mathsf{pke}^\star$ is readily translated to a pair for the pke experiment, so that the challenge ciphertext β received in return can be used to create the challenge ciphertext (α, β) under the scheme $\mathsf{pke}^\star$. The key to completing this reduction is emulating a decryption oracle for $\mathsf{pke}^\star$ on all ciphertexts other than a given challenge query (α, β), using access to a decryption oracle for pke which cannot be queried on the challenge query β. In this emulation, if a ciphertext (α', β) is submitted with $\alpha' \neq \alpha$, we shall return $\perp$; we need to argue that this would be the same result in an actual decryption too. This is because $\mathsf{pke}^\star.\mathsf{decrypt}$ first decrypts $\beta = \mathsf{pke.encrypt}(\mathsf{PK}, \mathsf{c}||\mathsf{d}||m)$ using SK (derived from $\mathsf{SK}^\star$) and obtains $\mathsf{c} = \alpha$; since $\alpha' \neq \alpha$, the decryption will output $\perp$.

Building on the above, we go on to prove $\mathsf{Anon\text{-}CCA}$ security for $\mathsf{pke}^\star$. Here, we have a couple of extra issues to take care of in the reduction. Firstly, since we do not know which of two public-keys are being used, we cannot translate a pair of messages for encryption under $\mathsf{pke}^\star$ to a pair for encryption under pke as above (as the message that gets encrypted under pke depends on the public key). Secondly, we need to consider the effect of decrypting a challenge ciphertext β produced using one key, using the secret-key corresponding to the other key.

To address these issues, first consider a hybrid experiment in which we build a reduction which knows the public-key PK_b being used for producing the challenge ciphertext. The first issue does not arise in this case. For emulating the decryption oracle, we follow the same strategy as before, even when the decryption oracle for the "wrong" key $\mathsf{SK}^\star_{1-b}$ is queried: that is, (α', β) will be decrypted as $\perp$. To justify this we shall argue that if $\mathsf{pke.decrypt}(\mathsf{SK}_{1-b}, \beta) = \widetilde{\alpha}||\widetilde{\mathsf{d}}||\widetilde{m}$ then $\widetilde{\alpha} = \alpha'$ only with negligible probability. First consider a hybrid in which the challenge ciphertext is (α, β) where $\beta = \mathsf{pke.encrypt}(\mathsf{PK}_b, m_{\text{dummy}})$, $m_{\text{dummy}} \in \mathcal{M}$ being a fixed dummy message. By CCA security of pke, this will not change the outcome of the $\mathsf{Anon\text{-}CCA}$ experiment for $\mathsf{pke}^\star$. Now, recall that $\mathsf{SK}^\star_{1-b} = (r_{\mathsf{pke}}, r_{\mathsf{com}})$ and $\mathsf{PK}^\star_{1-b}$ contains a commitment to a deterministic function of r_{pke}, using randomness r_{com}. Then, $\mathsf{PK}^\star_{1-b}$ has high min-entropy even conditioned on r_{pke} (this follows from the hiding property of com; alternately, one could simply require $\mathsf{com.Commit}$ to include $\omega(\log \kappa)$ random bits in its output). Also, with the above modification, β is now independent of r_{com}, and so is $\mathsf{pke.decrypt}(\mathsf{SK}_{1-b}, \beta)$. However, $\mathsf{pke}^\star.\mathsf{decrypt}(\mathsf{SK}^\star_{1-b}, (\alpha', \beta)) \neq \perp$ only if $\widetilde{\alpha}$ which is part of $\mathsf{pke.decrypt}(\mathsf{SK}_{1-b}, \beta)$ equals α'. Since $\widetilde{\alpha}$ determines $\mathsf{PK}^\star_{1-b}$ it too has high minentropy when r_{com} is chosen randomly, and since $\mathsf{pke.decrypt}(\mathsf{SK}_{1-b}, \beta)$ is independent of r_{com} (in the modified experiment), we conclude that the probability of this occurring is negligible.

Now we have a reduction which knows the the public-key PK_b being used for producing the challenge ciphertext, but does not query either decryption oracle on the challenge ciphertext β. Hence, if we change β to be the encryption of an arbitrary message, the outcome of the experiment will change at most negligibly. Finally, in this modified experiment, the reduction does not need to use the bit b, thereby giving a valid $\mathsf{Anon\text{-}CCA}$ adversary against pke with almost the same advantage as the adversary given against $\mathsf{pke}^\star$.

F Proof Omitted from Section 4.2

In this section we prove Theorem 1, by showing that the PKE scheme presented in Figure 1 is COA secure.

First, we describe how key reliability is achieved. Before accepting a secret-key $\mathsf{SK}^\star$, the accepting algorithm checks if the secret-key SK for pke contained in $\mathsf{SK}^\star$ is a "good" key: i.e., whether it has probability at least $1-p$ of correctly decrypting the encryption of a *random* message, for a small enough p (say $p = 1/10$), by running κ tests. If SK has a lower probability of correctness, then except with negligible probability, at least one of the tests will fail. Later, during encryption using the corresponding public key $\mathsf{PK}^\star$, t pairs of (individually) random messages are encrypted. The probability that a pair fails – i.e., at least one ciphertext in the pair fails to decrypt correctly – is bounded by $2p$, by the union bound. Since each pair is independently encrypted and decrypted, with high probability, the number of pairs that fail is not more than say $4tp$. In particular the probability that $t/2$ or more pairs fail is exponentially small in t, by Chernoff bound. Note that if more than $t/2$ pairs correctly decrypt, then the entire decryption succeeds.

Given this guarantee of key reliability (for accepted keys), the rest of existential consistency follows in a way similar to that of the previous construction. However, achieving $\mathsf{Anon\text{-}CCA}$ is much more complicated in this construction. We briefly sketch the main new idea needed for proving this. Here, for pke, we consider a variant of the $\mathsf{Anon\text{-}CCA}$ security game which allows the adversary to obtain a number of ciphertexts by submitting a vector of pairs of messages, rather than a single pair, for encryption (a secret bit b is used to choose encrypting the first message in every pair or the second message in every pair, using respectively the first or second of a pair of public-keys that the experiment picked). As before, the main task in the reduction of $\mathsf{Anon\text{-}CCA}$ security of $\mathsf{pke}^\star$ to $\mathsf{Anon\text{-}CCA}$ security of pke is to emulate a decryption oracle for $\mathsf{pke}^\star$ using a decryption oracle for pke. To answer a decryption query with a ciphertext $\mathsf{CT}^\star$, we need to carry out the decryption of several ciphertext pairs $\{\mathsf{CT}_0^i, \mathsf{CT}_1^i\}$ contained in $\mathsf{CT}^\star$. If none of these ciphertexts are part of the set of challenge ciphertexts we received (included in the challenge ciphertext we gave the adversary), then we can simply forward all of them to the external oracles for decryption. However, if some of them were challenge ciphertexts, we consider each of their decryption to be $\perp$, and carry out the rest of $\mathsf{pke}^\star.\mathsf{decrypt}$ using this.

To see that this is a sound emulation strategy, we consider a few cases. First consider the case that a decryption query $\mathsf{CT}^\star = (\xi, \tau)$ (not exactly equal to the challenge ciphertext) is made to the oracle that uses the same key as the one behind the challenge ciphertexts. By the unforgeability of the signature we may assume that $\xi \neq \xi^*$ where the challenge ciphertext is (ξ^*, τ^*). For the pke ciphertexts that occur in ξ as well as ξ^*, with high probability, they will be decrypted correctly. Now, if in a pair $(\mathsf{CT}_0^i, \mathsf{CT}_1^i)$ one of them was part of ξ^*, CCA security implies that it is improbable that the decryption of the two will XOR together to include c as part of it, where c is part of ξ; hence the pair can be decrypted as $\perp$ without altering the outcome of decrypting (ξ, τ) using $\mathsf{pke}^\star.\mathsf{decrypt}$. If the entire pair $(\mathsf{CT}_0^i, \mathsf{CT}_1^i)$ was part of ξ^*, then they will decrypt to include the same verification key verk as in the challenge ciphertext, and assuming that the signature scheme cannot be forged with respect to this key, cannot be part of the set S identified during decryption; thus again replacing the decryption of the pair with $\perp$ does not affect the outcome of the full decryption of (ξ, τ).

Next we consider the case when the decryption query is made to the key not used to produce the challenge ciphertext. It is in this case that we rely on the additional strings included in the public-key. Recall that in the previous scheme, we could consider a hybrid where it was information theoretically unlikely that the decryption using the wrong key will match the commitment c included in the ciphertext. However, this argument breaks down when using secret-sharing, as the adversary may be able to *predict* (though not control) what the outcome of decryption under the key would be; but since this decrypted value is only a share, the adversary can control the reconstructed value by choosing the other share appropriately. To solve this issue we shall require each share to individually match a high min-entropy tag that is included in the public-key. However, we cannot simply require the decryption of each ciphertext to contain the same string, because then the different pairs are correlated, and we cannot apply the concentration bounds we applied to argue that more than $t/2$ pairs will decode correctly with high probability. Hence we require each ciphertext CT_b^i to match a string r_b^i that is included in the public-key. Then as before, we can argue that the probability that a ciphertext honestly generated using one key (for a fixed message, by moving to an indistinguishable hybrid) will decrypt to include a certain independently sampled string is negligible.

G Ciphertext Resistance

Below, we prove Lemma 1 which states that any non-trivial COA-secure encryption scheme is ciphertext-resistant. For this first, we state and prove another lemma.

Lemma 2. *Suppose* $\mathsf{pke} = (\mathsf{skGen}, \mathsf{pkGen}, \mathsf{encrypt}, \mathsf{decrypt})$ *is a non-trivial COA-secure encryption scheme. Then, the following probabilities are negligible:*

$$\max_{SK_0 \in \mathcal{SK}} \Pr_{SK \leftarrow \mathsf{skGen}} [\mathsf{accept}(SK_0) = \textsc{sk} \ \wedge \ \mathsf{pkGen}(SK) = \mathsf{pkGen}(SK_0)] \tag{1}$$

$$\max_{PK_0 \in \mathcal{PK}} \Pr_{SK \leftarrow \mathsf{skGen}} [\mathsf{accept}(PK_0) = \textsc{pk} \ \wedge \ \mathsf{pkGen}(SK) = PK_0] \tag{2}$$

$$\max_{CT_0 \in \mathcal{CT}} \Pr_{SK \leftarrow \mathsf{skGen}} [\mathsf{accept}(CT_0) = \textsc{ct} \ \wedge \ \mathsf{decrypt}(SK, CT_0) \neq \bot] \tag{3}$$

$$\max_{\substack{PK_0 \in \mathcal{PK} \\ m \in \mathcal{M}}} \Pr_{SK \leftarrow \mathsf{skGen}} [\mathsf{accept}(PK_0) = \textsc{pk} \ \wedge \ \mathsf{decrypt}(SK, \mathsf{encrypt}(PK_0, m)) \neq \bot] \tag{4}$$

Proof. Note that (1) is upper bounded by (2). The latter can be shown to be negligible because of (semantic) security and existential consistency: Firstly, note that correctness is implied by the consistency requirements on accept, $\mathsf{decrypt}$ and $\mathsf{encrypt}/\mathsf{msgId}$: i.e.,

$$\delta := \max_{m \in \mathcal{M}} \Pr_{SK \leftarrow \mathsf{skGen}} [\mathsf{decrypt}(SK, \mathsf{encrypt}(\mathsf{pkGen}(SK), m)) \neq m]$$

is negligible. Now, suppose for some $PK_0 \in \mathcal{PK}$, $\Pr_{SK \leftarrow \mathsf{skGen}}[\mathsf{pkGen}(SK) = PK_0] = \epsilon$; then an adversary $\mathcal{A}$ who sends distinct m_0, m_1 (recall that $|\mathcal{M}| > 1$) to $\mathsf{Expt}_{\mathcal{A}}^{\mathsf{ano\text{-}cca}}(\kappa)$ will be able to predict b correctly with probability $\frac{1}{2} + \frac{\epsilon}{2} - \delta$, simply by checking if the public-key given to it equals PK_0 and if so decrypting the challenge ciphertext with some fixed secret-key SK_0 such that $\mathsf{pkGen}(SK_0) = PK_0$. Hence, by semantic security, ϵ must be negligible.

To upper bound (3), note that for any $CT_0 \in \mathcal{CT}$, by the previous part, $\Pr_{SK \leftarrow \mathsf{skGen}}[\mathsf{pkGen}(SK) = \mathsf{pke}^\star.\mathsf{pkId}(CT_0)]$ is negligible. However, by existential consistency, whenever $\mathsf{pkGen}(SK) \neq \mathsf{pke}^\star.\mathsf{pkId}(CT_0)$ we have that $\mathsf{decrypt}(SK, CT_0) = \bot$.

Finally, to upper bounded (4), note that if $\mathsf{pkId}(\mathsf{encrypt}(PK_0, m)) \in \{PK_0, \bot\}$ and $\mathsf{pkGen}(SK_0) \neq PK_0$, then $\mathsf{decrypt}(SK, \mathsf{encrypt}(PK_0, m)) = \bot$. This implies that (4) is upper bounded by the sum of (2) and a negligible quantity $\nu(\kappa)$ from Definition 2.

Proof (of Lemma 1). Lemma 1 follows from $\mathsf{Anon\text{-}CCA}$ security and Lemma 2. Consider a hybrid experiment in which the adversary is given oracle access to encryption and decryption using not the actual (SK, PK) pair, but a separate, independently generated key pair. Firstly, by $\mathsf{Anon\text{-}CCA}$ security the two hybrids are indistinguishable. Secondly, in this hybrid, using the bound on (3) from Lemma 1, ciphertext resistance holds. Hence ciphertext resistance holds in the original experiment too.

H Impossibility of Γ_{ppt}-IND-PRE Secure Encryption

Γ_{ppt} refers to the class of all probabilistic polynomial time Tests. In [2] it was pointed out that obfuscation does not have a Γ_{ppt}-IND-PRE secure implementation. Here we point out that in the same model as in [2] (i.e., without our extension), public-key encryption – and even symmetric-key encryption – cannot have a Γ_{ppt}-IND-PRE secure implementation.

Consider a schema $\boldsymbol{\Sigma}_{\mathsf{ske}}$ that allows creating a key agent, changing it into a ciphertext agent for a given message, and running a session with a ciphertext agent and its key agent to recover the message. Suppose we are given a candidate scheme Π_{ske} that purportedly is a Γ_{ppt}-IND-PRE secure implementation of $\boldsymbol{\Sigma}_{\mathsf{ske}}$. Let $\ell(\kappa)$ be an upperbound on the key-length in this scheme.

Then we consider $\mathsf{Test} \in \Gamma_{\mathsf{ppt}}$ that behaves as follows. After initialization Test uniformly picks $m \leftarrow \{0,1\}^{\ell(\kappa)+\kappa}$, encodes it into one or more messages and creates (and automatically transfers to User) handles for ciphertext agents for all these messages. Then it expects User to send back a (polynomially long) program σ. After receiving σ, Test transfers the key agent. Next, it expects user to send back an $\ell(\kappa)$ bit input x for σ. If x is such that $\sigma(x) = m$, Test sends the test-bit b to the User and otherwise it halts.

In the ideal world, User cannot access m until after it sends σ, and hence information-theoretically it is unlikely that $\sigma(x) = m$, since $|m| \gg |x|$. However, in the real execution with Π_{ske}, an adversary can set σ to be a program which will take as input a decryption key, and use it to decrypt the ciphertexts that the adversary received in the first step (which are hardwired into σ). Further, on receiving the decryption key, the adversary sends it to Test as x, so that, by the requisite correctness properties of Π_{ske}, with high probability, $\sigma(x) = m$ and learns b exactly. This violates indistinguishability preservation.

Extensions. In the above attack, Test is hiding even against a computationally unbounded adversary in the ideal world. As such, the impossibility extends to the weaker definition of Γ_{ppt}-s-IND-PRE as well.

Also note that simulation-based security implies Γ_{ppt}-IND-PRE security (and the weaker security of unbounded simulation implies Γ_{ppt}-s-IND-PRE security). Hence the above attack rules out (unbounded) simulation-based security for SKE if the decryption key can be transferred.

I Details in the Proof of $\boldsymbol{\Delta}$-IND-PRE Security

I.1 Extended Schema

An extended schema $\boldsymbol{\Sigma}_{\mathsf{ext}} = (\boldsymbol{\Sigma}, \boldsymbol{\Sigma}^{\dagger}_{\Pi_{\mathsf{pke}}})$ consists of the agent(s) of $\boldsymbol{\Sigma}$ and an additional agent $P^{\dagger}$, which behaves as follows. When $P^{\dagger}$ is run with an empty work-tape and input obj, it records $(\mathtt{obj}, \mathit{obj})$ on its work-tape, where $\mathtt{obj}$ is a keyword. When an agent is run in a session with a non-empty work-tape, first it sends the contents of its work-tapes and inputs to the first agent; then, if it is the first agent in the session, it computes $((\mathit{obj}'_1, y_1), \cdots, (\mathit{obj}'_t, y_t)) = \Pi_{\mathsf{pke}}.\mathsf{run}((\mathit{obj}_1, x_1), \cdots, (\mathit{obj}_t, x_t))$ where (obj_i, x_i) is received from the i^{th} agent, and (obj'_i, y_i) is returned to it. Then each agent updates its work-tape with obj'_i and outputs y_i.[21]

I.2 Notation

Before describing the simulators $\mathcal{S}^{\dagger}_b, \mathcal{S}^{\ddagger}$ and $\mathcal{S}^*$, we present some notation that we will use through out the proof. Consider the ideal execution involving $\mathsf{Test} \in \Delta$, $\mathcal{B}[\boldsymbol{\Sigma}_{\mathsf{pke}}]$ (or $\mathcal{B}[\boldsymbol{\Sigma}_{\mathsf{ext}}]$) and an adversary (simulator). Recall that being in Δ, Test would report every command it sends to $\mathcal{B}[\boldsymbol{\Sigma}_{\mathsf{pke}}]$ also to the adversary, except for transfer commands, for which it reports a pair of handles. We use variables like h and g to denote generic handles, or specifically sk, pk and ct to indicate handles of specific types. For clarity, we consider the handle-space for Test and Adv as disjoint sets $\widehat{\mathcal{H}}$ and $\overline{\mathcal{H}}$. We shall handles in $\widehat{\mathcal{H}}$ generically by variables like $\widehat{\mathsf{h}}$ and $\widehat{\mathsf{g}}$, or specifically as $\widehat{\mathit{sk}}$, $\widehat{\mathit{pk}}$ and $\widehat{\mathit{ct}}$, depending on the type of the handle; similarly, handles in $\overline{\mathcal{H}}$ are denoted by $\overline{\mathsf{h}}$, $\overline{\mathsf{g}}$, $\overline{\mathit{sk}}$, $\overline{\mathit{pk}}$ and $\overline{\mathit{ct}}$. Let $\mathtt{Type} : \widehat{\mathcal{H}} \cup \overline{\mathcal{H}} \to \{\mathtt{sk}, \mathtt{pk}, \mathtt{ct}\}$, denote the function mapping handles to their types.[22]

Below, we define several relations and functions over the handle-space $\widehat{\mathcal{H}} \cup \overline{\mathcal{H}}$, with respect to a history of interactions among an ideal adversary simulator, a $\mathsf{Test} \in \Delta$ and the schema (which can be $\mathcal{B}[\boldsymbol{\Sigma}_{\mathsf{ext}}]$ or $\mathcal{B}[\boldsymbol{\Sigma}_{\mathsf{pke}}]$). If h resulted from the command init to $\mathcal{B}[\boldsymbol{\Sigma}_{\mathsf{pke}}]$ or $\mathcal{B}[\boldsymbol{\Sigma}_{\mathsf{ext}}]$, then we write $\circ \xrightarrow{\mathsf{init}} \mathsf{h}$. We write $\mathit{sk} \xrightarrow{\mathtt{pkGen}} \mathit{pk}$ if pk is a handle that was returned from a session $(\mathit{sk}, \mathtt{pkGen})$. We write $\mathit{pk} \xrightarrow{\mathtt{encr}(m)} \mathit{ct}$ if ct is a handle that

[21] The adversary should be allowed to inspect the worktape of an agent of $\boldsymbol{\Sigma}^{\dagger}$ (possibly created by Test by operations on agents initialized by the adversary). To model this one could add a command (which, w.l.o.g., only the adversary will use) which prompts an agent in $\boldsymbol{\Sigma}^{\dagger}$ to output its work-tape contents. However, when working with tests in Δ, this facility is redundant. So we omit this from the description of $\boldsymbol{\Sigma}^{\dagger}$.

[22] When interacting with $\mathsf{Test} \in \Delta$, an adversary can keep track of $\mathtt{Type}(\widehat{\mathsf{h}})$ for each handle $\widehat{\mathsf{h}}$ received by Test, using the information reported to it by Test. This is because each session results in an *a priori* fixed number of handles of known types, and the handles are picked deterministically (sequentially) from within the handle-space. Also, on receiving a handle $\overline{\mathsf{h}}$ from $\mathcal{B}[\boldsymbol{\Sigma}_{\mathsf{pke}}]$, the adversary can find out its type from $\mathcal{B}[\boldsymbol{\Sigma}_{\mathsf{pke}}]$.

was returned from a session $(\mathsf{pk}, (\mathtt{encr}, m))$. Also, we write, $\mathsf{sk} \xrightarrow{\mathsf{clone}} \mathsf{sk}'$ if sk' was returned from a session $(\mathsf{sk}, \mathtt{clone})$. Finally, $\overline{h} \xrightarrow{\mathsf{transfer}} \widehat{g}$ denotes that Test received $\widehat{g}$ when User transferred $\overline{h}$; also, $(\widehat{h}_0, \widehat{h}_1) \xrightarrow{\mathsf{transfer}} \overline{g}$ denotes the fact that Test reported $(\mathsf{transfer}, \widehat{h}_0, \widehat{h}_1)$ to User (recall the behavior of $\mathsf{Test} \in \varDelta$) and User received $\overline{g}$ as a result of the transfer.

We define relations $\overset{b}{\equiv}$ and $\overset{b}{\rightsquigarrow}$ among handles in $\widehat{\mathcal{H}} \cup \overline{\mathcal{H}}$, for $b \in \{0,1\}$, as follows. For $h, g \in \widehat{\mathcal{H}} \cup \overline{\mathcal{H}}$, we say that $h \overset{b}{\equiv} g$ if, from the point of view of the ideal adversary, assuming that the test-bit given to Test equals b, h and g were both obtained by transfers of the same handle; further, for two public-key handles $\mathsf{pk}, \mathsf{pk}'$ we let $\mathsf{pk} \overset{b}{\equiv} \mathsf{pk}'$ if they were derived from secret-key handles $\mathsf{sk}, \mathsf{sk}'$ that were either obtained through cloning or transfers of the same handle.

More formally, we define the functions $\mathtt{parent}_b : \widehat{\mathcal{H}} \cup \overline{\mathcal{H}} \to \widehat{\mathcal{H}} \cup \overline{\mathcal{H}} \cup \{\perp\}$, for $b \in \{0,1\}$ as follows.

$$\mathtt{parent}_b(g) = \begin{cases} h & \text{if } h \xrightarrow{\mathsf{transfer}} g \\ h & \text{if } (h_0, h_1) \xrightarrow{\mathsf{transfer}} g \text{ s.t. } h = h_b \\ \perp & \text{otherwise (i.e., } g \text{ was not the result of a transfer)} \end{cases}$$

Also, we define $\mathtt{root}_b(g)$ to be the unique "ancestor" h of g such that $\mathtt{parent}_b(h) = \perp$. That is, $\mathtt{root}_b(g)$ is the root of the tree containing g in the forest defined by the function $\mathtt{parent}_b$. For secret-key handles, we define another similar function, incorporating cloning:

$$\mathtt{parent}^+_b(\mathsf{sk}) = \begin{cases} \mathsf{sk}' & \text{if } \mathtt{parent}(\mathsf{sk}) = \mathsf{sk}' \\ \mathsf{sk}' & \text{if } \mathsf{sk}' \xrightarrow{\mathsf{clone}} \mathsf{sk} \\ \perp & \text{otherwise (i.e., } \mathsf{sk} \text{ was not the result of a transfer or cloning)} \end{cases}$$

Also, let $\mathtt{root}^+_b(\mathsf{sk})$ be the root of the tree containing sk in the forest defined by the function $\mathtt{parent}^+_b$.

We say $g \overset{b}{\equiv} g'$ if one of the following conditions hold.

- $\mathtt{root}_b(g) = \mathtt{root}_b(g')$
- $\exists \mathsf{pk}, \mathsf{pk}', \mathsf{sk}, \mathsf{sk}'$ s.t. $\mathsf{pk} \overset{b}{\equiv} g$ and $\mathsf{pk}' \overset{b}{\equiv} g'$ (by the above rule), $\mathsf{sk} \xrightarrow{\mathsf{pkGen}} \mathsf{pk}$, $\mathsf{sk}' \xrightarrow{\mathsf{pkGen}} \mathsf{pk}'$, and $\mathtt{root}^+_b(\mathsf{sk}) = \mathtt{root}^+_b(\mathsf{sk}')$.

We define $h \overset{b}{\rightsquigarrow} g$ to indicate that the handle g was *derived from an original handle* h. That is, $h \overset{b}{\rightsquigarrow} g$ if $\circ \xrightarrow{\mathsf{init}} h$, and one of the following conditions holds:

- $h \overset{b}{\equiv} g$,
- $\exists \mathsf{pk}', \mathsf{sk}'$ s.t. $h \overset{b}{\equiv} \mathsf{sk}'$, $\mathsf{sk}' \xrightarrow{\mathsf{pkGen}} \mathsf{pk}'$ and $g \overset{b}{\equiv} \mathsf{pk}'$, or
- $\exists \mathsf{ct}, \mathsf{pk}, m$ s.t. $h \overset{b}{\rightsquigarrow} \mathsf{pk}$ (as above) and $\mathsf{pk} \xrightarrow{\mathsf{encr}(m)} \mathsf{ct}$ and $g \overset{b}{\equiv} \mathsf{ct}$.

Note that any handle g will have at most one handle h such that $h \overset{b}{\rightsquigarrow} g$ (in fact, unless $\exists g'$ such that $g' \xrightarrow{\mathsf{clone}} g$, g will have exactly one h such that $h \overset{b}{\rightsquigarrow} g$).

I.3 Description of the Simulators

Our proof of security makes use of simulators $\mathcal{S}^{\ddagger}$, $\mathcal{S}^{\dagger}_b$ and $\mathcal{S}^*$ that are closely related to each other. In order to simulate the view of Adv in the ideal world, a simulator must process the handles of the form $\overline{\mathsf{sk}}$, $\overline{\mathsf{pk}}$ and $\overline{\mathsf{ct}}$ (corresponding to secret key, public key, and ciphertext *agents*) transferred by Test, as well as the (possibly malformed) objects Adv attempts to transfer to Test.

All the simulators will maintain the history of its interaction with the schema and Test, in order to keep track of the relations $\overset{b}{\equiv}$ and $\overset{b}{\rightsquigarrow}$. They maintain a simulated repository $\mathsf{R}^{\mathsf{User}}$ with entries of the form $(\overline{h}, \mathsf{obj})$

for all the handles $\overline{h}$ received from the schema, and corresponding objects. The simulators also maintain simulated repositories $\mathsf{R}^{\mathsf{Test}}$ with entries of the form $(\widehat{h}, \mathit{obj})$ for handles on the side of Test. In the case of $\mathcal{S}^{\ddagger}$, instead of a single $\mathsf{R}^{\mathsf{Test}}$, a pair $(\mathsf{R}_0^{\mathsf{Test}}, \mathsf{R}_1^{\mathsf{Test}})$ will be maintained, with a lazy strategy for filling in the *obj* part for the entries. Further, we shall consider lists T_b and L_b for $b \in \{0,1\}$, which will be helpful in analyzing the correctness of the simulation.

All the simulators internally run Adv and interact with Test and the schema. We describe the behavior of the simulators in terms of how they process the various messages from Test, Adv and schema.

- All simulators let Adv and Test directly interact with each other, except for the transfer of objects and reports from Test regarding commands it sends to the schema.
- $\mathcal{S}_b^{\dagger}$ and $\mathcal{S}^{\ddagger}$ process the objects transferred by Adv identically (but $\mathcal{S}^{\ddagger}$ updates both $\mathsf{R}_0^{\mathsf{Test}}$ and $\mathsf{R}_1^{\mathsf{Test}}$, whereas $\mathcal{S}_b^{\dagger}$ updates $\mathsf{R}^{\mathsf{Test}}$), by using the schema $\boldsymbol{\Sigma}^{\dagger}_{\Pi_{\mathsf{pke}}}$ as described in Figure 9. $\mathcal{S}^{*}$ is a computationally unbounded wrapper over $\mathcal{S}^{\ddagger}$, that simulates access to $\boldsymbol{\Sigma}^{\dagger}_{\Pi_{\mathsf{pke}}}$, using only access to the schema $\boldsymbol{\Sigma}_{\mathsf{pke}}$ as described in Figure 13.
- $\mathcal{S}_b^{\dagger}$ processes handles transferred by Test as well as the reports from Test using a perfect simulation of $\mathcal{I}[\Pi, \mathsf{Repo}_{\mathsf{Test}}]$ *assuming that* Test *has test-bit set to* b, as described in Figure 10. $\mathcal{S}^{\ddagger}$ uses a procedure that does not rely on knowing the test-bit, as described in Figure 11. The core function resolveObject used by $\mathcal{S}^{\ddagger}$ to assign objects to handles is described in Figure 12.

$\mathcal{S}^{\ddagger}$ assigns objects to handles in a lazy fashion using the function resolveObject. When the execution of a command $(\mathsf{transfer}, \widehat{h}_0, \widehat{h}_1)$ is to be simulated, $\mathcal{S}^{\ddagger}$ uses resolveObject to compute an object *obj* and sends it to the adversary; also for each $b \in \{0,1\}$, the entry $(\widehat{h}_b, \mathit{obj})$ is added to $\mathsf{R}_b^{\mathsf{Test}}$. The lists T_0 and T_1 contain *tentative* objects that are assigned to handles while a transferred object is computed. If a command $(\mathsf{transfer}, \widehat{h}_0, \widehat{h}_1)$ is to be simulated and for some b if already $(\widehat{h}_b, \mathit{obj}) \in T_b$, then the object transferred should be *obj*. If it turns out that it will reveal the test-bit if a handle in T_b is to be transferred corresponding to test-bit b, then that handle is "locked" by moving it to the list L_b. For a hiding Test, a locked handle will need to be transferred only with negligible probability.

$\mathcal{S}_b^{\dagger}$ and $\mathcal{S}^{\ddagger}$: Processing objects transferred by Adv

When Adv attempts to transfer object *obj* to Test:

- If there is an entry $(\overline{h}, \mathit{obj}) \in \mathsf{R}^{\mathsf{User}}$, then send $(\mathsf{transfer}, \overline{h})$ to $\mathcal{B}[\boldsymbol{\Sigma}_{\mathsf{ext}}]$.
- Else, if some $(\overline{sk}, SK) \in \mathsf{R}^{\mathsf{User}}$ satisfies one of the following conditions, proceed as follows. (If there are multiple such entries in $\mathsf{R}^{\mathsf{User}}$, pick one arbitrarily).
  - If $\mathit{obj} = SK^{\star}||\mathit{pad}$ such that $\mathsf{pke}^{\star}.\mathsf{pkGen}(SK) = \mathsf{pke}^{\star}.\mathsf{pkGen}(SK^{\star})$, then send $(\mathsf{run}, (\overline{sk}, \mathtt{clone}))$ to $\mathcal{B}[\boldsymbol{\Sigma}_{\mathsf{ext}}]$ to obtain a new handle $\overline{sk}'$. Add $(\overline{sk}', \mathit{obj})$ to $\mathsf{R}^{\mathsf{User}}$ and transfer $\overline{sk}'$.
  - If $\mathsf{pke}^{\star}.\mathsf{pkGen}(SK) = \mathit{obj}$, then send $(\mathsf{run}, (\overline{sk}, \mathtt{pkGen}))$ to $\mathcal{B}[\boldsymbol{\Sigma}_{\mathsf{ext}}]$ to obtain a new handle $\overline{pk}$. Add $(\overline{pk}, \mathit{obj})$ to $\mathsf{R}^{\mathsf{User}}$ and transfer $\overline{pk}$.
  - Else, if $\mathsf{pke}^{\star}.\mathsf{decrypt}(SK, \mathit{obj}) = m \neq \perp$, then obtain $\overline{pk}$ as above and send $(\mathsf{run}, (\overline{pk}, (\mathtt{encr}, m)))$ to $\mathcal{B}[\boldsymbol{\Sigma}_{\mathsf{ext}}]$ to obtain a new handle $\overline{ct}$. Add $(\overline{ct}, \mathit{obj})$ to $\mathsf{R}^{\mathsf{User}}$ and transfer $\overline{ct}$.
- Else, if $\Pi_{\mathsf{pke}}.\mathsf{receive}(\mathit{obj}) \neq \perp$, then send $(\mathsf{init}, P^{\dagger}, \mathit{obj})$ to $\mathcal{B}[\boldsymbol{\Sigma}_{\mathsf{ext}}]$ (where $P^{\dagger}$ is the single agent in $\boldsymbol{\Sigma}^{\dagger}_{\Pi_{\mathsf{pke}}}$), and obtain a handle $\overline{h}$ in return; then, add $(\overline{h}, \mathit{obj})$ to $\mathsf{R}^{\mathsf{User}}$ and send $(\mathsf{transfer}, \overline{h})$ to $\mathcal{B}[\boldsymbol{\Sigma}_{\mathsf{ext}}]$.
- In all of the above cases, let $\widehat{h}$ be the handle to be returned to Test by $\mathcal{B}[\boldsymbol{\Sigma}_{\mathsf{ext}}]$ for this transfer. $\mathcal{S}_b^{\dagger}$ adds $(\widehat{h}, \mathit{obj})$ to $\mathsf{R}^{\mathsf{Test}}$, while $\mathcal{S}^{\ddagger}$ adds $(\widehat{h}, \mathit{obj})$ to both $\mathsf{R}_0^{\mathsf{Test}}$ and $\mathsf{R}_1^{\mathsf{Test}}$.

Fig. 9 Simulators $\mathcal{S}_b^{\dagger}$ and $\mathcal{S}^{\ddagger}$: Objects Transferred by Adv

$\mathcal{S}_b^\dagger$ processing commands by Test

Test reports all its commands to the user. Process them as follows:

- On receiving a report of $(\mathsf{init}, P, \kappa)$ from Test, let $\mathit{SK} \leftarrow \mathsf{pke}^\star.\mathsf{skGen}(\kappa)$ and add the record $(\widehat{h}, \mathit{SK})$ to lists $\mathsf{R}^{\mathsf{Test}}$, where $\widehat{h}$ is the next handle to be delivered to Test.
- On receiving a report of $(\mathsf{run}, (\widehat{h}_1, x_1))$ from Test which could result in a new object (i.e., x_1 is of the form `pkGen` or $(\mathtt{encr}, m)$), look up the entry $(\widehat{h}_1, \mathit{obj}_1)$ in $\mathsf{R}^{\mathsf{Test}}$, execute $\Pi.\mathsf{run}((\mathit{obj}_1, x_1))$, and if it produces a new object obj (not $\perp$), then record $(\widehat{h}, \mathit{obj})$ in $\mathsf{R}^{\mathsf{Test}}$, where $\widehat{h}$ is the next handle to be delivered to Test.
- On receiving a report $(\mathsf{transfer}, \widehat{h}_0, \widehat{h}_1)$ from Test and a handle $\overline{h}$ from schema, look up the (unique) entry $(\widehat{h}_b, \mathit{obj}) \in \mathsf{R}^{\mathsf{Test}}$, add $(\overline{h}, \mathit{obj})$ to $\mathsf{R}^{\mathsf{User}}$, and send obj to Adv.

Fig. 10 Simulator $\mathcal{S}_b^\dagger$: Processing commands by Test

$\mathcal{S}^\ddagger$ processing commands by Test

- Reports from Test of commands other than transfer are used to maintain the relations required by resolveObject.
- On receiving a report $(\mathsf{transfer}, \widehat{h}_0, \widehat{h}_1)$ from Test and a handle $\overline{h}$ from schema, proceed as follows:
  - Invoke the resolveObject function to obtain a tuple (obj, α) or $\perp$. If $\perp$, abort.
  - Else, for $b = 0$ and $b = 1$, add $(\widehat{g}, \mathit{obj})$ to $\mathsf{R}_b^{\mathsf{Test}}$ for every $\widehat{g} \overset{b}{\equiv} \widehat{h}_b$ (unless the entries already exist).
  - Let $\overline{h}$ be the next handle for User; add $(\overline{h}, \mathit{obj})$ to $\mathsf{R}^{\mathsf{User}}$. If $\alpha = (\widehat{sk}_0, \widehat{sk}_1, \mathit{SK})$, then, for $b = 0, 1$, for all $\widehat{g} \overset{b}{\equiv} \widehat{sk}_b$, add $(\widehat{g}, \mathit{SK})$ to T_b; if $\alpha = (\widehat{sk}_0, \widehat{sk}_1)$, then for $b = 0, 1$, for all $\widehat{g} \overset{b}{\equiv} \widehat{sk}_b$, add $\widehat{g}$ to L_b.
  - Finally, send obj to Adv.

Fig. 11 $\mathcal{S}^\ddagger$ processing commands by Test (see Figure 12 for subroutine resolveObject).

Function $\mathsf{resolveObject}$ **used by** $\mathcal{S}^{\ddagger}$

This function is invoked on receiving a report $(\mathsf{transfer}, \widehat{h}_0, \widehat{h}_1)$ from Test. Returns $\perp$ (for aborting) or (obj, α), where obj is the object to be sent to the adversary, and α contains optional information to update T_b and L_b lists.

- Return $\perp$ if one of the following conditions hold:
  - $\mathtt{Type}(\widehat{h}_0) \neq \mathtt{Type}(\widehat{h}_1)$.
  - $\exists$ handle $\overline{g}$ in $\mathsf{R}^{\mathsf{User}}$, and $b \in \{0,1\}$, such that $\overline{g} \overset{b}{\rightsquigarrow} \widehat{h}_b$ but $\overline{g} \overset{1-b}{\not\rightsquigarrow} \widehat{h}_{1-b}$.
  - $\exists b \in \{0,1\}, \mathit{obj}$ s.t. $(\widehat{h}_b, \mathit{obj}) \in \mathsf{R}_b^{\mathsf{Test}}$ but $(\widehat{h}_{1-b}, \mathit{obj}) \notin \mathsf{R}_{1-b}^{\mathsf{Test}}$.
  - $\widehat{h}_0 \in L_0$ or $\widehat{h}_1 \in L_1$.
- Else, if $\exists \mathit{obj}$, $\forall b \in \{0,1\}$, $(\widehat{h}_b, \mathit{obj}) \in \mathsf{R}_b^{\mathsf{Test}}$, then return $(\mathit{obj}, \perp)$.
- Otherwise, proceed as follows depending on $\mathtt{Type}(\widehat{h}_0) = \mathtt{Type}(\widehat{h}_1)$.

Case $\mathtt{Type}(\widehat{h}_0) = \mathtt{Type}(\widehat{h}_1) = \mathtt{sk}$.

- If $\forall b \in \{0,1\}$ $\nexists \mathit{obj}(\widehat{h}_b, \mathit{obj}) \in \mathsf{R}_b^{\mathsf{Test}} \cup T_b$, then let $\mathit{SK} \leftarrow \mathsf{pke}^{\star}.\mathsf{skGen}$ and return $(\mathit{SK}, \perp)$.
- Else, return $\perp$.

Case $\mathtt{Type}(\widehat{h}_0) = \mathtt{Type}(\widehat{h}_1) = \mathtt{pk}$.

- If $\exists(\overline{\mathit{sk}}, \mathit{SK}) \in \mathsf{R}^{\mathsf{User}}$ s.t. $\forall b \in \{0,1\}$, $\overline{\mathit{sk}} \overset{b}{\rightsquigarrow} \widehat{h}_b$, let $\mathit{PK} \leftarrow \mathsf{pke}^{\star}.\mathsf{pkGen}(\mathit{SK})$ and return $(\mathit{PK}, \perp)$.
- Else, if $\exists \widehat{\mathit{sk}}_0, \widehat{\mathit{sk}}_1$ s.t. $\widehat{\mathit{sk}}_0 \overset{0}{\rightsquigarrow} \widehat{h}_0$, $\widehat{\mathit{sk}}_1 \overset{1}{\rightsquigarrow} \widehat{h}_1$, and:
  - If $\exists \mathit{SK}, b \in \{0,1\}$ s.t. $(\widehat{\mathit{sk}}_b, \mathit{SK}) \in \mathsf{R}_b^{\mathsf{Test}}$ but $(\widehat{\mathit{sk}}_{1-b}, \mathit{SK}) \notin \mathsf{R}_{1-b}^{\mathsf{Test}}$, then return $\perp$.
  - Else, if $\exists \mathit{SK}, \forall b \in \{0,1\}(\widehat{\mathit{sk}}_b, \mathit{SK}) \in \mathsf{R}_b^{\mathsf{Test}} \cup T_b$, let $\mathit{PK} \leftarrow \mathsf{pke}^{\star}.\mathsf{pkGen}(\mathit{SK})$ and return $(\mathit{PK}, \perp)$.
  - Else, if $\forall b \in \{0,1\}$ $\nexists \mathit{SK}(\widehat{\mathit{sk}}_b, \mathit{SK}) \in \mathsf{R}_b^{\mathsf{Test}} \cup T_b$, then let $\mathit{SK} \leftarrow \mathsf{pke}^{\star}.\mathsf{skGen}$, $\mathit{PK} \leftarrow \mathsf{pke}^{\star}.\mathsf{pkGen}(\mathit{SK})$, $\alpha = (\widehat{\mathit{sk}}_0, \widehat{\mathit{sk}}_1, \mathit{SK})$ and return (PK, α).
  - Else, if $\exists \mathit{SK}, b \in \{0,1\}$ s.t. $(\widehat{\mathit{sk}}_b, \mathit{SK}) \in T_b$ but $(\widehat{\mathit{sk}}_{1-b}, \mathit{SK}) \notin T_{1-b}$, then then let $\mathit{SK} \leftarrow \mathsf{pke}^{\star}.\mathsf{skGen}$, $\mathit{PK} \leftarrow \mathsf{pke}^{\star}.\mathsf{pkGen}(\mathit{SK})$, $\alpha = (\widehat{\mathit{sk}}_0, \widehat{\mathit{sk}}_1)$ (corresponding to adding them to L_0 and L_1) and return (PK, α).
- If none of the above conditions hold, return $\perp$.

Case $\mathtt{Type}(\widehat{h}_0) = \mathtt{Type}(\widehat{h}_1) = (\mathtt{ct}, \ell)$.

- If $\exists \underline{g}_0, \underline{g}_1$ s.t. $\forall b \in \{0,1\}$, $\underline{g}_b \underset{m}{\overset{b}{\rightsquigarrow}} \widehat{h}_b$, and $\mathsf{length}(m_0) = \mathsf{length}(m_1)$, then return $\perp$.
- If $\exists(\overline{g}, \mathit{obj}) \in \mathsf{R}^{\mathsf{User}}$ s.t. $\forall b \in \{0,1\}$, $\overline{\mathit{sk}} \underset{m_b}{\overset{b}{\rightsquigarrow}} \widehat{h}_b$, such that $m_0 \neq m_1$, return $\perp$.
- If $\exists m, (\overline{\mathit{sk}}, \mathit{SK}) \in \mathsf{R}^{\mathsf{User}}$ s.t. $\forall b \in \{0,1\}$, $\overline{\mathit{sk}} \underset{m}{\overset{b}{\rightsquigarrow}} \widehat{h}_b$, then let $\mathit{PK} = \mathsf{pke}^{\star}.\mathsf{pkGen}(\mathit{SK})$, $\mathit{CT} \leftarrow \mathsf{pke}^{\star}.\mathsf{encrypt}(m, \mathit{PK})$ and return $(\mathit{CT}, \perp)$.
- If $\exists m, (\overline{\mathit{pk}}, \mathit{PK}) \in \mathsf{R}^{\mathsf{User}}$ s.t. $\forall b \in \{0,1\}$, $\overline{\mathit{pk}} \underset{m}{\overset{b}{\rightsquigarrow}} \widehat{h}_b$, then let $\mathit{CT} \leftarrow \mathsf{pke}^{\star}.\mathsf{encrypt}(m, \mathit{PK})$ and return $(\mathit{CT}, \perp)$.
- Else, if $\exists \widehat{\mathit{sk}}_0, \widehat{\mathit{sk}}_1$ s.t. $\widehat{\mathit{sk}}_0 \underset{m_0}{\overset{0}{\rightsquigarrow}} \widehat{h}_0$, $\widehat{\mathit{sk}}_1 \underset{m_1}{\overset{1}{\rightsquigarrow}} \widehat{h}_1$, and:
  - If $\exists \mathit{SK}, b \in \{0,1\}$ s.t. $(\widehat{\mathit{sk}}_b, \mathit{SK}) \in \mathsf{R}_b^{\mathsf{Test}}$ but $(\widehat{\mathit{sk}}_{1-b}, \mathit{SK}) \notin \mathsf{R}_{1-b}^{\mathsf{Test}}$, then return $\perp$.
  - Else, if $m_0 = m_1 =: m$ and $\exists \mathit{SK}, \forall b \in \{0,1\}(\widehat{\mathit{sk}}_b, \mathit{SK}) \in \mathsf{R}_b^{\mathsf{Test}} \cup T_b$, let $\mathit{PK} \leftarrow \mathsf{pke}^{\star}.\mathsf{pkGen}(\mathit{SK})$, $\mathit{CT} \leftarrow \mathsf{pke}^{\star}.\mathsf{encrypt}(m, \mathit{PK})$ and return $(\mathit{CT}, \perp)$.
  - Else, if $m_0 = m_1 =: m$ and $\forall b \in \{0,1\}$ $\nexists \mathit{SK}(\widehat{\mathit{sk}}_b, \mathit{SK}) \in \mathsf{R}_b^{\mathsf{Test}} \cup T_b$, then let $\mathit{SK} \leftarrow \mathsf{pke}^{\star}.\mathsf{skGen}$, $\mathit{PK} \leftarrow \mathsf{pke}^{\star}.\mathsf{pkGen}(\mathit{SK})$, $\mathit{CT} \leftarrow \mathsf{pke}^{\star}.\mathsf{encrypt}(m, \mathit{PK})$, $\alpha = (\widehat{\mathit{sk}}_0, \widehat{\mathit{sk}}_1, \mathit{SK})$ and return (CT, α).
  - $\star$ Else, if $m_0 \neq m_1$ or $\exists \mathit{SK}, b \in \{0,1\}$ s.t. $(\widehat{\mathit{sk}}_b, \mathit{SK}) \in T_b$ but $(\widehat{\mathit{sk}}_{1-b}, \mathit{SK}) \notin T_{1-b}$, then then let $\mathit{SK} \leftarrow \mathsf{pke}^{\star}.\mathsf{skGen}$, $\mathit{PK} \leftarrow \mathsf{pke}^{\star}.\mathsf{pkGen}(\mathit{SK})$, $\mathit{CT} \leftarrow \mathsf{pke}^{\star}.\mathsf{encrypt}(m, \mathit{PK})$, where m is an arbitrary message with $\mathsf{length}(m) = \ell$, $\alpha = (\widehat{\mathit{sk}}_0, \widehat{\mathit{sk}}_1)$ (corresponding to adding them to L_0 and L_1) and return (CT, α).
- If none of the above conditions hold, return $\perp$.

Fig. 12 Core function used by $\mathcal{S}^{\ddagger}$ for processing handles transferred by Test

$\mathcal{S}^*$ (as a wrapper over a $\mathcal{B}[\boldsymbol{\Sigma}_{\mathsf{ext}}]$-adversary $\mathcal{S}^\ddagger$)

$\mathcal{S}^*$ interacts with $\mathcal{B}[\boldsymbol{\Sigma}_{\mathsf{pke}}]$, while simulating to $\mathcal{S}^\ddagger$ the interface to $\mathcal{B}[\boldsymbol{\Sigma}_{\mathsf{ext}}]$, using super-polynomial computational power. It maintains two tables, Z_1 to map handles received from $\mathcal{B}[\boldsymbol{\Sigma}_{\mathsf{pke}}]$ (denoted as $\widetilde{h}$ etc.) to objects and Z_2 to map them to handles that it sends to $\mathcal{S}^\ddagger$ (denoted as $\overline{h}$ etc.). The following subroutines are used by $\mathcal{S}^*$ to interact with $\mathcal{B}[\boldsymbol{\Sigma}_{\mathsf{pke}}]$, and to read and update Z_1.

Subroutine $\mathsf{makeSK}(\mathit{obj})$
Precondition: $\mathit{obj} = \mathit{obj}_0 || \mathit{pad}$, where $\mathit{pad} \in \{0,1\}^\kappa$ and $\mathsf{pke}^\star.\mathsf{accept}(\mathit{obj}_0) = \textsc{sk}$, or $\mathit{obj} = \bot$
If $\exists \widetilde{sk}$ s.t. $(\widetilde{sk}, \mathit{obj}) \in Z_1$, then return $\widetilde{sk}$. Else, if $\mathit{obj} = SK^\star || \mathit{pad}$ such that there is an entry $(\widetilde{sk}_0, SK) \in Z_1$ with $\mathsf{pke}^\star.\mathsf{pkGen}(SK^\star) = \mathsf{pke}^\star.\mathsf{pkGen}(SK)$, then send $(\mathsf{run}, (\widetilde{sk}_0, \texttt{clone}))$ to $\mathcal{B}[\boldsymbol{\Sigma}_{\mathsf{pke}}]$ (if multiple matching entries are found, an arbitrary one can be used); else, send init to $\mathcal{B}[\boldsymbol{\Sigma}_{\mathsf{pke}}]$. Let $\widetilde{sk}_1$ be the handle received in return. If $\mathit{obj} \neq \bot$, add $(\widetilde{sk}_1, SK^\star)$ to Z_1. Return $\widetilde{sk}_1$.

Subroutine $\mathsf{makePK}(\mathit{obj})$
Precondition: $\mathsf{pke}^\star.\mathsf{accept}(\mathit{obj}) = \textsc{pk}$ or $\mathit{obj} = \bot$
If $\exists \widetilde{pk}$ s.t. $(\widetilde{pk}, \mathit{obj}) \in Z_1$, then return $\widetilde{pk}$. Else, let $SK := \mathsf{pke}^\star.\mathsf{skId}(\mathit{obj})$ and $\widetilde{sk} := \mathsf{makeSK}(SK)$, and send $(\mathsf{run}, (\widetilde{sk}, \texttt{pkGen}))$ to $\mathcal{B}[\boldsymbol{\Sigma}_{\mathsf{pke}}]$. Let $\widetilde{pk}$ be the handle received in return. If $\mathit{obj} \neq \bot$, add $(\widetilde{pk}, \mathit{obj})$ to Z_1. Return $\widetilde{pk}$.

Subroutine $\mathsf{makeCT}(\mathit{obj})$
Precondition: $\mathsf{pke}^\star.\mathsf{accept}(\mathit{obj}) = \textsc{ct}$
If $\exists \widetilde{ct}$ s.t. $(\widetilde{ct}, \mathit{obj}) \in Z_1$, then return $\widetilde{ct}$. Else, let $m = \mathsf{pke}^\star.\mathsf{msgId}(\mathit{obj})$, $PK := \mathsf{pke}^\star.\mathsf{pkId}(\mathit{obj})$ and $\widetilde{pk} := \mathsf{makePK}(PK)$, and send $(\mathsf{run}, (\widetilde{pk}, (\texttt{encr}, m)))$ to $\mathcal{B}[\boldsymbol{\Sigma}_{\mathsf{pke}}]$. Let $\widetilde{ct}$ be the handle received in return. Add $(\widetilde{ct}, \mathit{obj})$ to Z_1. Return $\widetilde{ct}$.

Commands from $\mathcal{S}^\ddagger$ to $\mathcal{B}[\boldsymbol{\Sigma}_{\mathsf{ext}}]$:

- When $\mathcal{S}^\ddagger$ sends a command $(\mathsf{init}, P^\dagger, \mathit{obj})$ (i.e., an init command for an agent in $\boldsymbol{\Sigma}^\dagger_{\Pi_{\mathsf{pke}}}$) to $\mathcal{B}[\boldsymbol{\Sigma}_{\mathsf{ext}}]$, depending on whether obj belongs to $\mathcal{SK}^\star, \mathcal{PK}^\star$ or $\mathcal{CT}^\star$, proceed as follows (if $\mathit{obj} \notin \mathcal{SK}^\star \cup \mathcal{PK}^\star \cup \mathcal{CT}^\star$, send $\bot$ as the response to $\mathcal{S}^\ddagger$):
  - If $\mathit{obj} = \mathit{obj}_0 || \mathit{pad}$, where $\mathit{pad} \in \{0,1\}^\kappa$ and $\mathsf{pke}^\star.\mathsf{accept}(\mathit{obj}_0) = \textsc{sk}$, let $\widetilde{h} \leftarrow \mathsf{makeSK}(\mathit{obj})$.
  - If $\mathsf{pke}^\star.\mathsf{accept}(\mathit{obj}) = \textsc{pk}$ let $\widetilde{h} \leftarrow \mathsf{makePK}(\mathit{obj})$.
  - If $\mathsf{pke}^\star.\mathsf{accept}(\mathit{obj}) = \textsc{ct}$ let $\widetilde{h} \leftarrow \mathsf{makeCT}(\mathit{obj})$.

  In all the above cases, add $(\widetilde{h}, \overline{h})$ to Z_2 where $\overline{h}$ denotes the next handle to be returned by $\mathcal{B}[\boldsymbol{\Sigma}_{\mathsf{ext}}]$ (being simulated). Send $\overline{h}$ to $\mathcal{S}^\ddagger$.
- When $\mathcal{S}^\ddagger$ sends any other init, run or $\mathsf{transfer}$ command to $\mathcal{B}[\boldsymbol{\Sigma}_{\mathsf{ext}}]$, $\mathcal{S}^*$ simply relays the command to $\mathcal{B}[\boldsymbol{\Sigma}_{\mathsf{pke}}]$, but substituting each handle $\overline{h}$ in the command with $\widetilde{h}$ using the Z_2 map. The response from $\mathcal{B}[\boldsymbol{\Sigma}_{\mathsf{pke}}]$ is relayed back to $\mathcal{S}^\ddagger$, but after replacing each new handle $\widetilde{h}$ in the response with a new handle $\overline{h}$ (i.e., the next handle to be returned by $\mathcal{B}[\boldsymbol{\Sigma}_{\mathsf{ext}}]$), and adding an entry $(\widetilde{h}, \overline{h})$ to Z_2. (If a handle in the response is $\bot$, indicating that the agent halted, it is not replaced with a new handle, but is kept as $\bot$.)

Transfers from Test*:* When $\mathcal{B}[\boldsymbol{\Sigma}_{\mathsf{pke}}]$ delivers a handle $\widetilde{h}$, $\mathcal{S}^*$ will deliver a new handle $\overline{h}$ to give to $\mathcal{S}^\ddagger$ and makes an entry $(\widetilde{h}, \overline{h})$ in Z_2.

Fig. 13 Simulator $\mathcal{S}^*$

I.4 Indistinguishability of Hybrids.

In this section, we sketch the argument that consecutive hybrids are indistinguishable.

$\mathsf{H}_0 \approx \mathsf{H}_1$. The difference between the two hybrids is that $\mathcal{I}[\Pi, \mathsf{Repo}_{\mathsf{Test}}]$ in H_0 is replaced by $(\mathcal{B}[\boldsymbol{\Sigma}_{\mathsf{ext}}], \mathcal{S}_0^{\dagger})$ in H_1, where $\mathcal{S}_0^{\dagger}$ uses the information sent by $\mathsf{Test} \in \Delta$ to simulate $\mathcal{I}[\Pi, \mathsf{Repo}_{\mathsf{Test}}]$ internally. But the two hybrids can differ in how honestly generated objects (by Test or $\mathcal{S}_0^{\dagger}$, respectively in H_0 and H_1) interact with objects generated by the adversary. In particular, in H_1 sessions which mix handles from the two sub-schemas of $\mathcal{B}[\boldsymbol{\Sigma}_{\mathsf{ext}}]$ will result in halting of the agents, whereas in H_0, this depends on how the objects interact with each other in an algorithm for Π_{pke}. Another reason for the two hybrids to differ is that in H_1 independently generated tags ($\mathit{sk\text{-}tag}$, $\mathit{pk\text{-}tag}$ or $\mathit{ct\text{-}tag}$) can collide. But we can argue that these events have negligible probability.

Formally, we *couple* the executions of H_0 and H_1 by considering a single experiment which runs both the executions using a common random-tape, such that on marginalizing each execution by itself is correctly distributed. The randomness used by $\mathcal{I}[\Pi, \mathsf{Repo}_{\mathsf{Test}}]$ for operations of Π_{pke} in H_0 are identified with the randomness used by $\mathcal{S}_0^{\dagger}$ in H_1. The randomness used in H_1 by $\mathcal{B}[\boldsymbol{\Sigma}_{\mathsf{ext}}]$ to sample the tags ($\mathit{sk\text{-}tag}$, $\mathit{pk\text{-}tag}$ or $\mathit{ct\text{-}tag}$) are not used in H_0. The random-tapes of the adversary and Test are the same in both parts of the coupled execution. Then we specify the following "bad events" in the coupled execution. Here we say that a secret-key SK *matches* an object obj if $\mathsf{pke}^{\star}.\mathsf{pkGen}(\mathit{SK}) = \mathsf{pke}^{\star}.\mathsf{pkGen}(\mathit{obj})$ or $\mathsf{pke}^{\star}.\mathsf{pkGen}(\mathit{SK}) = \mathit{obj}$ or $\mathsf{decrypt}(\mathit{SK}, \mathit{obj}) \neq \bot$.

1. In H_1, initializing a new $\mathcal{B}[\boldsymbol{\Sigma}_{\mathsf{pke}}]$ agent results in the same $\mathit{sk\text{-}tag}$, $\mathit{pk\text{-}tag}$ or $\mathit{ct\text{-}tag}$ as in a previous initialization.
2. In H_0, a secret-key SK generated by $\mathcal{I}[\Pi, \mathsf{Repo}_{\mathsf{Test}}]$, such that its public-key (or a secret-key generating that public-key) was not transferred to the user, matches a new object obj transferred by the user to Test.
3. In H_0, a secret-key SK generated by $\mathcal{I}[\Pi, \mathsf{Repo}_{\mathsf{Test}}]$ matches an object obj derived (using $\mathsf{pke}^{\star}.\mathsf{pkGen}$ or $\mathsf{pke}^{\star}.\mathsf{encrypt}$) by $\mathcal{I}[\Pi, \mathsf{Repo}_{\mathsf{Test}}]$ from an object obj_0 transferred by the user such that obj_0 itself did not match any secret-key of $\mathcal{I}[\Pi, \mathsf{Repo}_{\mathsf{Test}}]$ when it was transferred.
4. In H_0, a secret-key SK_0 transferred by the user matches an obj derived by $\mathcal{I}[\Pi, \mathsf{Repo}_{\mathsf{Test}}]$ from a secret-key SK that it generated such that when SK_0 was transferred it did not match any secret-key of $\mathcal{I}[\Pi, \mathsf{Repo}_{\mathsf{Test}}]$.
5. In H_0, an object obj derived by $\mathcal{I}[\Pi, \mathsf{Repo}_{\mathsf{Test}}]$ from a secret-key SK that it generated *equals* an object obj_1 derived from an object obj_0 transferred by the user to Test, such that obj_0 did not match any secret-key of $\mathcal{I}[\Pi, \mathsf{Repo}_{\mathsf{Test}}]$ when it was transferred.

We say that a coupled execution does not diverge if the view of the adversary and Test is identical in both the H_0 and H_1 parts. Conditioned on the bad events not occurring, we claim that a coupled execution does not diverge. This is verified inductively, over each message from the adversary or Test. Indeed, as long as there have been no divergence previously, the objects sent to the adversary – created by $\mathcal{I}[\Pi, \mathsf{Repo}_{\mathsf{Test}}]$ or $\mathcal{S}_0^{\dagger}$ – are identical in H_0 and H_1. Also, the sessions run by Test involving only handles that are for $P^{\dagger}$ agents in H_1 (i.e., the agents of the extended schema which carry objects themselves) will be executed by $\mathcal{B}[\boldsymbol{\Sigma}_{\mathsf{ext}}]$ in exactly the same way as $\mathcal{I}[\Pi, \mathsf{Repo}_{\mathsf{Test}}]$ executes the objects received from the adversary. For the sessions that, in H_1, only have handles for P agents (i.e., agents of the schema $\mathcal{B}[\boldsymbol{\Sigma}_{\mathsf{pke}}]$, including possibly cloned secret-key agents), as long as the first bad event above has not occurred, existential consistency[23] ensures that Test receives the same outcome as from those sessions in H_0. Finally, in H_1 sessions involving both $P^{\dagger}$ and P handles (decryption or comparison) will result in both the agents halting and producing $\bot$ outputs; in H_0 this is ensured if the remaining bad events do not occur.

To complete the proof we note that both bad events have negligible probability in the respective hybrids. This follows for the first event simply due to the exponentially large space for tags; for the second event this follows from ciphertext resistance, and for the other events it follows from Lemma 2.

[23] We do not rely on full-fledged existential consistency at this point, since the secret-keys transferred by the adversary are not replaced by ideal keys in H_1 (unless they are clones of a key originally transferred by Test); only the honestly generated keys transferred by Test and their clones transferred back by the adversary are replaced. Public-keys and ciphertexts interact with secret-keys and public-keys derived from them (in decryption or comparison) deterministically, and hence are faithfully simulated.

$\mathsf{H}_1 \approx \mathsf{H}_2$. Two prove this, we introduce an intermediate hybrid $\mathsf{H}_{1|2}$ that uses a simulator $\mathcal{S}_0^{\ddagger}$ (instead of $\mathcal{S}_0^{\dagger}$ or $\mathcal{S}^{\ddagger}$). $\mathcal{S}_0^{\ddagger}$ behaves identically as $\mathcal{S}^{\ddagger}$, except in the step in $\mathsf{resolveObject}$ marked with a $\star$ in Figure 12. Here, $\mathcal{S}_0^{\ddagger}$ prepares CT differently as follows: if $\exists \mathit{SK}$ s.t. $(\widehat{\mathit{sk}}_0, \mathit{SK}) \in T_0$ it uses this SK (instead of sampling $\mathit{SK} \leftarrow \mathsf{pke}^{\star}.\mathsf{skGen}$) and it obtains CT by encrypting m_0 (instead of an arbitrary m with $\mathsf{length}(m) = \ell$).

Firstly, we argue that $\mathsf{H}_{1|2} \approx \mathsf{H}_2$ by $\mathsf{Anon\text{-}CCA}$ security of $\mathsf{pke}^{\star}$. For this the two hybrids can be coupled such that they differ only in how certain ciphertexts are generated (from one of two possible keys and messages), and the corresponding secret-keys are never transferred to the adversary (as the corresponding handles will be included in the lists L_0, L_1, and if Test transfers any of them, the execution is aborted without transferring the key to the adversary).

Secondly, $\mathcal{S}_0^{\ddagger}$ is in fact a *lazy version* of $\mathcal{S}_0^{\dagger}$ which assigns objects to Test's handles only when they are transferred. But, like $\mathcal{S}^{\ddagger}$, it aborts the execution if $\mathsf{resolveObject}$ returns $\perp$. The executions of H_1 and $\mathsf{H}_{1|2}$ can be coupled such that, conditioned on $\mathcal{S}_0^{\ddagger}$ not aborting, the two executions are *identical*.

Finally, we argue that the probability of $\mathsf{H}_{1|2}$ aborting is negligible, so that we can conclude that $\mathsf{H}_1 \approx \mathsf{H}_{1|2}$ (and hence, $\mathsf{H}_1 \approx \mathsf{H}_2$). For this, it is enough to argue that the probability of H_2 aborting is negligible. It can be seen that when an event that leads to abort occurs, $\mathcal{S}^{\ddagger}$ can infer the test bit unambiguously (with a few additional queries to $\mathcal{B}[\boldsymbol{\Sigma}_{\mathsf{ext}}]$). Hence, if ϵ_2 and ϵ_5 respectively denote the probabilities of H_2 and H_5 aborting, then an adversary derived from $\mathcal{S}^{\ddagger}$ and $\mathcal{A}$ will have an advantage of $(\epsilon_2 + \epsilon_5)/2$ in distinguishing the two hybrids. Since $\mathsf{H}_2 \approx \mathsf{H}_5$, we conclude that ϵ_2 is negligible, as required.

$\mathsf{H}_2 \approx \mathsf{H}_3$. As before, we *couple* the executions of H_2 and H_3: a single experiment runs both the executions using a common random-tape for Test, $\mathcal{A}$ and operations of Π_{pke} (carried out by $\boldsymbol{\Sigma}^{\dagger}_{\Pi_{\mathsf{pke}}}$ in H_2 and $\mathcal{S}^*$ in H_3) that are common to both hybrids, such that on marginalizing, each execution by itself is correctly distributed.

$\mathcal{S}^*$ uses $\mathsf{pke}^{\star}.\mathsf{pkId}$ and $\mathsf{pke}^{\star}.\mathsf{msgId}$ as guaranteed by COA security of $\mathsf{pke}^{\star}$, so that the two hybrids proceed identically from the point of view of Test and $(\mathcal{S}^{\ddagger}, \mathcal{A})$, barring when one of the following events happen.

1. In H_2, $\boldsymbol{\Sigma}^{\dagger}_{\Pi_{\mathsf{pke}}}$ carries out an encryption $\mathsf{pke}^{\star}.\mathsf{encrypt}(\mathit{PK}, m)$ to obtain a ciphertext $\mathit{CT}^{\star}$ and subsequently a decryption $\mathsf{pke}^{\star}.\mathsf{decrypt}(\mathit{SK}, \mathit{CT}^{\star})$ which results in an output other than m, while $\mathsf{pke}^{\star}.\mathsf{pkGen}(\mathit{SK}) = \mathit{PK}$.
2. In H_3, initializing a new $\mathcal{B}[\boldsymbol{\Sigma}_{\mathsf{pke}}]$ agent results in the same *sk-tag*, *pk-tag* or *ct-tag* as in a previous initialization.

Note that $\boldsymbol{\Sigma}^{\dagger}_{\Pi_{\mathsf{pke}}}$ works on only PK and SK that have been accepted by accept. Hence we can apply the existential consistency guarantees to show that the first event happens only with negligible probability. The second event has negligible probability due to the length of the tags. Thus $\mathsf{H}_2 \approx \mathsf{H}_3$.