Published in the *Proceedings of the 16th Annual Information Security Conference*, pp. 49–65

Encrypted Secret Sharing and Analysis by Plaintext Randomization ⋆

Stephen R. Tate[1], Roopa Vishwanathan[1], and Scott Weeks[2]

[1] Department of Computer Science, University of North Carolina at Greensboro, Greensboro, NC 27402
[2] Department of Computer Science, George Mason University, Fairfax, VA 22030

Abstract. In this paper we consider the problem of secret sharing where shares are encrypted using a public-key encryption (PKE) scheme and ciphertexts are publicly available. While intuition tells us that the secret should be protected if the PKE is secure against chosen-ciphertext attacks (i.e., CCA-secure), formally proving this reveals some subtle and non-trivial challenges. We isolate the problems that this raises, and devise a new analysis technique called "plaintext randomization" that can successfully overcome these challenges, resulting in the desired proof. The encryption of different shares can use one key or multiple keys, with natural applications in both scenarios.

1 Introduction

During the past three decades, cryptography research has been very successful in developing clear notions of security and rigorous techniques for reasoning about security of cryptographic primitives and protocols. Formal notions of security in cryptography have evolved in essentially two main directions, with reduction-based proofs developing from the initial work of Goldwasser and Micali [7] and simulation-based proofs from the initial work of Goldreich, Micali, and Wigderson [6]. While we have a good understanding of how to reason about security in these settings, there are recurring issues with composability: using one secure protocol as a component of another protocol while retaining security inside the higher-level protocol. Somewhat counter-intuitively, some protocols (e.g., some zero-knowledge proofs) fail to maintain security even when multiple copies of the same protocol are run concurrently [5].

In this paper, we explore a combination of public-key encryption (PKE) with secret sharing, and in the process develop a general-purpose proof technique for analysis of cryptographic schemes and protocols that use public-key encryption (PKE) as a component. Perhaps the simplest example of such a system is the common practice of hybrid encryption: doing large scale encryption by first encrypting a random session key using a PKE scheme, and then using that session key with a symmetric cipher for bulk encryption. Due to the inefficiency of public

⋆ This material is based upon work supported by the National Science Foundation under Grant No. 0915735.

key encryption, hybrid encryption has been standard practice since the 1980s, and our intuition tells us that if the PKE scheme and the symmetric cipher are both secure in some sense (e.g., against chosen ciphertext attacks) then the combination of these two components into a hybrid system should also be secure. However, despite widespread use of hybrid encryption, the security of hybrid encryption was not rigorously established until the 2003 work of Cramer and Shoup [3]. A key insight in Cramer and Shoup's analysis was the introduction of the notion of a "key encapsulation mechanism" (KEM), which can be built from a CCA-secure PKE scheme. The value of KEM does not come from the power of this new cryptographic primitive, but rather comes from the clarity it brings to the *analysis* of hybrid encryption. In this paper, we focus on improving the analysis process from the beginning, so that we obtain clear proofs directly, with no need to introduce a new primitive such as a KEM. While hybrid encryption is a very simple example of this, our analysis technique can be applied to any protocol that uses a PKE scheme to hide secrets used within the protocol — we use the term "PKE-hybrid" to refer to protocols like this.

Unfortunately, current proof techniques are not sufficient for some PKE-hybrid problems, including the very practical problem of secret sharing with encrypted shares. To understand this problem, consider a standard key escrow situation, in which a company escrows copies of keys for the company's officers so that keys can be recovered if a certain number of board members agree. This is a classic threshold secret sharing situation, but in the real world having board members keep copies of shares of all officer's keys (which might change somewhat regularly) locally would not be practical. A better solution would be to have each board member maintain their own long-term key (perhaps on a smartcard), and have the company store encrypted shares of the escrowed keys, encrypted with board members keys, on a central server. This way shares can be updated without interaction of the board members, but board members would still be needed in order to decrypt the shares for a key recovery.

To understand why standard proof techniques do not work for this problem, consider a situation in which a CCA-secure PKE is used to encrypt shares from a perfect k-of-n threshold secret sharing scheme. We mirror a CCA game by creating a game in which we provide two secrets to the game oracle, which encrypts shares of one of these secrets for the adversary who must guess which secret is used. Using standard techniques we would try to simulate this adversary and game oracle in a CCA PKE game to relate to the security of the PKE scheme. However, we must make multiple encryptions and they must be consistent across the shares that are provided. Multiple consistent encryptions suggests using a multi-query oracle such as the left-right or real-or-random security notions of Bellare *et al.* [2]; however, we must allow the adversary to decrypt *some* of the encrypted shares, only disallowing decryption of a set that would allow reconstruction of the secret. These two properties, consistent encryptions and allowing some decryptions, are fundamentally opposed in standard techniques, making analysis of this situation particularly difficult.

1.1 Plaintext Randomization

To overcome the problems described in the previous section, we have developed a new analysis technique that we call *plaintext randomization*, which we successfully use to prove a strong security result for the general "secret-sharing with encrypted shares" problem. We believe this technique will be useful in the analysis of a wide variety of PKE-hybrid protocols.

Consider a cryptographic problem in which security is defined using a game between an adversary and a game oracle — we don't make any assumptions about the goal of the adversary, so that it does not need to be a CCA-style distinguishability goal, but rather can be *any* adversary goal that is well-defined. The game oracle makes some use of a PKE scheme, but this is internal to the game and not something that the adversary necessarily sees. The adversary may in fact have no direct access to the PKE scheme at all, including key generation, encryption, or decryption, but only sees the effects of these operations as allowed by the game oracle definition. Internally, all access to the PKE scheme by the game oracle goes through a generic interface that neither depends on the precise PKE scheme being used nor gives access to randomness used internally to the PKE functions.

Plaintext randomization then is the following technique: internal to the game oracle, every time the PKE encryption function is called as $E_{PK}(p)$ (for plaintext p and public key PK) we replace the plaintext with a random string r of the same length and instead call $c = E_{PK}(r)$, using ciphertext c in place of what would have been the encrypted plaintext. To allow for consistent decryption, the game oracle remembers these "encryptions" by storing pairs (c, p) so that if the decryption function is used by the game oracle on ciphertext c, the plaintext p will be returned rather than the actual decryption of c (which would give r). By modifying the game in this way, we remove any use of meaningful ciphertexts, and the storage of pairs (c, p) allows the game to provide restricted and consistent decryptions, solving the two problems that we identified for secret sharing with encrypted shares. Furthermore, the use of ciphertexts that are unrelated to actual plaintexts "cuts" any hybrid use of the PKE from the rest of the protocol, exactly the property we need to enable or simplify proofs for PKE-hybrid problems.

There are a few technicalities that must be addressed for this to work, such as the ability to randomly sample the plaintext space (defined as a "plaintext-samplable PKE" in Definition 3) and a restriction on how the game in question uses PKE secret keys (defined as "sk-oblivious" in Definition 4). Once these formalities are established, we are able to prove a result which we call the "Plaintext Randomization Lemma" that bounds the difference between the adversary's success probability in the original game and the success probability in the modified game that uses plaintext randomization. This is the key to the subsequent proof for secret sharing with encrypted shares.

Based on the high-level description given above, two questions about related work come to mind: Is this anything more than the well-known "real-or-random" security definition? And if multiple decryptions of a set of ciphertexts are allowed,

do selective opening attacks come into play? We address these two questions below.

Relation to Real-or-Random Security In real-or-random (ROR) security, an adversary is tasked with distinguishing between the encryption of provided plaintext and encryption of randomized plaintext [2]. Our technique replaces plaintexts with randomized plaintexts in the same manner as ROR security, but our model takes chosen-ciphertext security (i.e., ROR-CCA) one step further by adding the ability to consistently "decrypt" some ciphertexts, regardless of whether the real or random plaintext was encrypted.

Consider an attempt to reduce the encrypted secret sharing (ESS) problem to ROR security: an ESS adversary could be used to create an ROR adversary in which shares of a challenge value are encrypted with the ROR adversary, and the resulting ciphertexts provided to the ESS adversary which must decide whether they are shares of the challenge value or unrelated ciphertexts. This is a natural definition of ESS security, but it is inherent in the definition of ESS that the adversary should be allowed to decrypt *some* of these ciphertexts. However, if the ciphertexts come from ROR-CCA oracle queries, then we are necessarily disallowed from decrypting *any* of the ciphertexts produced in this way. Furthermore, the ESS adversary doesn't even know the "real" side of the "real-or-random" encryption, as the real values are embedded within the ESS game and not revealed to the ESS adversary except through specific, controlled decryption requests. Therefore, allowing for consistent decryptions is not something that is under the control of the adversary — it must be embedded in the ESS game itself, which is precisely what plaintext randomization does.

In the absence of any decryptions that must be made consistent, our plaintext randomization technique can be used simply as an abstraction for replacing PKE within a game, although the benefit is really notational in this case. In particular, if only PKE encryptions are performed (i.e., only chosen plaintext queries), then plaintext randomization really does behave the same as ROR-CPA security, and in such a case a direct reduction to real-or-random security might be more appropriate, depending on how the PKE is exposed to the adversary. The examples we consider in this paper all rely on CCA security, and hence make full use of the plaintext randomization technique.

Selective Opening Attacks The encrypted secret sharing problem provides a set of ciphertexts to the adversary and allows the adversary to open some subset of these ciphertexts. The underlying plaintexts are not independent since they are shares of a single secret, and this immediately raises a concern of selective opening attacks [4] and whether IND-CCA security is sufficient for the underlying PKE scheme. While this might be a concern for some applications, the encrypted secret sharing problem is based on trusted parties (either services or secure hardware such as a smartcard or trusted platform module) acting as both encryption and decryption oracles, where the randomness used in performing an encryption is never available external to the trusted party.

Our modular notation for PKE schemes ensures this property: Definition 2 defines the interface to a PKE scheme, and only encrypt/decrypt oracle calls are allowed, with no access to randomization used in encryption or revealed in decryption. Applications in which randomization might be revealed and made available to an adversary either directly or indirectly do not map to our definitions, and hence the plaintext randomization technique could not be applied to schemes in which selective opening attacks are a danger. The problems that we consider in this paper, natural and practical versions of encrypted secret sharing, *do* map to our definitions and selective opening is not an issue. We believe that many problems involving some type of escrow of secrets by trusted agents share these attributes.

1.2 Our Contributions

We briefly summarize the contributions of this paper below.

- We define notation that can be used in cryptographic games that use a PKE scheme inside the game oracle, for a uniform treatment and clear identification of how the PKE scheme is integrated into the protocol and game.
- We introduce the technique of *plaintext randomization* and prove the Plaintext Randomization Lemma, a powerful tool for analysis of PKE-hybrid systems.
- We formally define the "secret sharing with encrypted shares" problem, where shares are encrypted using public key encryption and a general key mapping function. We specify a PKE-hybrid scheme that uses a CCA-secure PKE in conjunction with a perfect secret sharing scheme, and prove the security of such a scheme.

While the original purpose of this work was to provide a security analysis for encrypted secret sharing, we believe that the plaintext randomization technique will be useful in a wide variety of situations, and will ease analysis of many protocols that make use of PKE as a component. As an example of this, we give a greatly simplified analysis (compared to prior work) of hybrid encryption.

2 Cryptographic Security and Games

All our schemes are parametrized by a security parameter λ. Specific parameters such as key sizes will depend on λ in operation-specific ways, and security is analyzed in terms of the probability of some event occurring as a function of λ — specifically, we want the probability of some bad event (related to a security compromise) to be a function that decreases as λ increases. For this notion, we use the standard notion of a negligible function.

Definition 1. *A function $f : Z \to R$ is **negligible in** λ (or just "negligible" when f is a function of a well-understood security parameter λ) if for every positive integer c_0 there exists an integer c_1 such that for all $\lambda > c_1$,*

$$|f(\lambda)| < \frac{1}{\lambda^{c_0}} \ .$$

If an event happens with probability $p(\lambda)$, we say that the event occurs "with overwhelming probability" if $p(\lambda) = 1 - f(\lambda)$, where $f(\lambda)$ is negligible.

Public-key cryptography is one of the foundations of modern cryptography, and is based on the idea that encryption and decryption can use different keys that are generated in pairs consisting of a public key (for encryption) and a secret key (for decryption). To formalize the idea of public-key encryption, we identify the three core operations that any public-key encryption scheme must provide, resulting in the following definition. Providing the full keypair, both public and secret keys, to the decryption function is slightly non-standard, but simplifies our presentation without changing the technical aspects.

Definition 2. *A **Public-Key Encryption (PKE)** scheme (referred to as "a PKE" for brevity) is defined by four sets and three probabilistic polynomial time operations. The sets are $\mathcal{PK}$, the set of public keys; $\mathcal{SK}$, the set of secret keys; $\mathcal{PT}$, the set of plaintexts; and $\mathcal{CT}$, the set of ciphertexts. The algorithms are the following:*

- *$KeyGen : 1^* \to \mathcal{PK} \times \mathcal{SK}$ — when called as $KeyGen(1^\lambda)$, where λ is a security parameter, produces a random public/secret pair (pk, sk) where $pk \in \mathcal{PK}$ and $sk \in \mathcal{SK}$.*
- *$Encrypt : \mathcal{PK} \times \mathcal{PT} \to \mathcal{CT}$ — when called as $Encrypt(pk, p)$, where $pk \in \mathcal{PK}$ and $p \in \mathcal{PT}$, produces ciphertext $c \in \mathcal{CT}$. It is not required that all plaintexts be valid for every public key, so we use $\mathcal{PT}(pk)$ to denote the set of valid plaintexts for a particular public key pk. If Encrypt is called with an invalid plaintext (i.e., $p \notin \mathcal{PT}(pk)$), then the operation fails and special value $\perp$ is returned.*
- *$Decrypt : \mathcal{PK} \times \mathcal{SK} \times \mathcal{CT} \to \mathcal{PT}$ — when called as $Decrypt(pk, sk, c)$, where $pk \in \mathcal{PK}$, $sk \in \mathcal{SK}$ and $c \in \mathcal{CT}$, produces plaintext $p \in \mathcal{PT}$. We can similarly restrict the ciphertext set to ciphertexts that are valid for a specific secret key sk, which we denote by $\mathcal{CT}(sk)$.*

We require that for any (pk, sk) produced by $KeyGen$, and for any plaintext $p \in \mathcal{PT}(pk)$, with overwhelming probability $Decrypt(pk, sk, Encrypt(pk, p)) = p$.

This definition provides the only way to interface with the PKE scheme, and so in particular neither the game oracle nor the adversary can have any access to randomization used during encryption. For the techniques described in this paper, we need PKEs that allow for random sampling from the set of valid plaintexts, a property of all widely-used PKE schemes. We call this property "plaintext-samplability," defined as follows.

Definition 3. *A **plaintext-samplable PKE** is a PKE scheme that, in addition to all operations of a standard PKE scheme, supports the following operation:*

- *$PTSample : \mathcal{PK} \times \mathcal{PT} \to \mathcal{PT}$ — when called as $PTSample(pk, p)$, where $pk \in \mathcal{PK}$ and $p \in \mathcal{PT}$, produces a random plaintext of the same length as a supplied plaintext $p \in \mathcal{PT}$. Specifically, x is uniformly chosen from $\{x \mid x \in \mathcal{PT}(pk)$ and $|x| = |p|\}$.*

In reduction-based security proofs, security of cryptosystems is defined in terms of a game between a probabilistic polynomial time (PPT) adversary and a game oracle. The oracle sets internal, persistent state variables, and answers queries for the adversary. The goal of the adversary is typically to determine some information in the internal state of the game oracle (e.g., a hidden bit) based just on oracle queries. A game G is described in terms of the interface to the game oracle, as a set of functions of the following form:

- $\mathsf{G.Initialize}(1^\lambda)$: Sets up persistent variables that are maintained throughout the game based on a security parameter λ. This can involve generating one or more random keys, and picking a random bit that the adversary will need to guess.
- $\mathsf{G}.\mathit{OracleQuery}(\cdots)$: One or more functions are defined that allow the adversary to query the oracle. These definitions are the heart of the security game.
- $\mathsf{G.IsWinner}(a)$: Takes a value a from the adversary at the end of the game, and returns true or false depending on whether a is a winning answer. This is always the "final answer" for the game, and once the adversary provides an answer a for $\mathsf{IsWinner}$ the game no longer accepts any more oracle queries.

A game typically relies on certain cryptographic operations, which we indicate by a superscript on the game name. For example, if some game G makes use of PKE scheme $\mathfrak{S}$, we can indicate this specific version of G by writing $\mathsf{G}^{\mathfrak{S}}$, and operations then might be denoted as $\mathsf{G}^{\mathfrak{S}}.\mathsf{Initialize}(1^\lambda)$, etc. Games that use PKE schemes typically do so in a generic way, treating keys as opaque objects. We are specifically interested in protocols that treat secret keys in this way, so introduce the following terminology.

Definition 4. *A game G that uses a PKE scheme $\mathfrak{S}$ is* **sk-*oblivious*** *if, for any keypair (pk, sk) produced by $\mathfrak{S}.KeyGen$, the only way that G uses sk is to pass sk back unmodified in calls to $\mathfrak{S}.Decrypt$. In such a situation, we can say that "G makes sk-oblivious use of $\mathfrak{S}$".*

The goal of the adversary is to win the game (i.e., produce an answer a such that $\mathsf{G.IsWinner}(a) = \mathsf{true}$) with higher probability than simply guessing an answer randomly. Typically the "answer" is a single hidden bit that needs to be guessed, although our definition purposely avoids making this a requirement so that other goals could be accommodated. In the case where the answer is a single bit, the probability that a random guess would give a winning answer is $\frac{1}{2}$, and the goal is to win with probability non-negligibly larger than $\frac{1}{2}$, leading to the definition of the "advantage" against a game.

Definition 5. *The "advantage" of an adversary A in game G is denoted $Adv_{A,\mathsf{G}}$, and defined as $Adv_{A,\mathsf{G}} = |P(\mathsf{G}.\mathit{IsWinner}(A^G(\lambda))) - \frac{1}{2}|$, where the probability is taken over both the random choices of A and the random choices of G . For some description of resource bounds B (e.g., time complexity, number of encryption or decryption queries, etc.), we also refer to the best possible advantage of any such*

adversary against a particular game, written $Adv_{\mathsf{G}}(B)$, defined as $Adv_{\mathsf{G}}(B) = \sup_A Adv_{A,\mathsf{G}}$, where the supremum is taken over all adversaries A that meet the bounds requirements B.

For a game G that defines the security of some type of cryptographic scheme, the goal is typically to find some specific scheme $\mathfrak{C}$ such that $Adv_{\mathsf{G}^{\mathfrak{C}}}$ is negligible.

2.1 Multi-User CCA Security

As an extension to the better-known single key CCA security model for PKE, Bellare *et al.* considered security of public-key encryption when the system is used with multiple users (i.e., multiple keypairs) and multiple challenges are made that are answered consistently [1]. This is sometimes referred to as multi-user "left-right" security, since every encryption challenge is made with a pair of values, and the encryption oracle consistently encrypts either the left or the right value throughout the game. A key property of this model is that since encryption pairs are chosen by the adversary, plaintexts of the provided ciphertexts can be related in arbitrary ways, opening up the possibility of complex attacks. The following describes the multi-user CCA security game $\mathsf{PK\text{-}MU}_n^{\mathfrak{S}}$ parametrized by the number of users n and a PKE scheme $\mathfrak{S}$.

- $\mathsf{PK\text{-}MU}_n^{\mathfrak{S}}.\mathsf{Initialize}(1^\lambda)$: For $i = 1$ to n, the oracle generates keypairs $(pk_i, sk_i) = \mathfrak{S}.KeyGen(1^\lambda)$, picks a random bit $b \in \{0, 1\}$, and sets C as an initially empty set of challenge ciphertexts. $pk_1, \cdots, pk_n$ are returned to the adversary.
- $\mathsf{PK\text{-}MU}_n^{\mathfrak{S}}.\mathsf{Decrypt}(i, x)$: If $(i, x) \in C$, the oracle returns $\perp$; otherwise, it returns $\mathfrak{S}.Decrypt(pk_i, sk_i, x)$.
- $\mathsf{PK\text{-}MU}_n^{\mathfrak{S}}.\mathsf{PEncrypt}(i, x_0, x_1)$: The oracle calculates $c = \mathfrak{S}.Encrypt(pk_i, x_b)$, adds (i, c) to C, and returns c to the adversary.
- $\mathsf{PK\text{-}MU}_n^{\mathfrak{S}}.\mathsf{IsWinner}(a)$: Takes a bit a from the adversary, and returns true if and only if $a = b$.

Note that $\mathsf{PEncrypt}$ ("pair encrypt") is similar to $\mathsf{Challenge}$ in the standard single-key CCA2 game, except that it can be called multiple times, and consistently encrypts either the first or second argument. While this is equivalent to the definition of Bellare *et al.*, we use a single multi-key oracle, as in the recent definition of Hofheinz and Jager [9]. For settings that only need the single-key version of CCA-security, we give simplified results in Appendix A.

3 Plaintext Randomization

In this section, we present the technique of plaintext randomization. This technique creates a PKE scheme that can maintain some internal state between encryption and decryption requests. This stateful behavior would not be very useful in most real-life cryptographic uses of public key encryption, since the state would need to be protected and shared between encryption and decryption

operations, which may be executed on separate systems. Nonetheless, it is possible to define such a stateful PKE scheme, and when this PKE is used inside a cryptographic game oracle — which can keep state secret from users of the oracle, and encryption and decryption are performed by the same party — it turns out to be a very useful concept.

Consider an adversary A that plays a game G that uses a plaintext-samplable PKE scheme $\mathfrak{S}$. We create a stateful PKE scheme that is based on $\mathfrak{S}$, but in which ciphertexts are meaningless, and cannot reveal any information about the corresponding plaintext other than perhaps its length. Specifically, ciphertexts are simply encryptions of random plaintexts of a given length, and the scheme uses internal state to keep track of plaintext/ciphertext pairs so that encryption followed by decryption gives the required result. We formalize this idea in the following definition.

Definition 6. *Given a plaintext-samplable PKE scheme $\mathfrak{S}$, the **plaintext randomization** of $\mathfrak{S}$ is a set of functions that acts as a PKE scheme, denoted $\mathfrak{S}$-rand, defined as follows:*

- $\mathfrak{S}$-rand.$KeyGen(1^\lambda)$ *computes* $(pk, sk) = \mathfrak{S}.KeyGen(1^\lambda)$ *and returns* (pk, sk).
- $\mathfrak{S}$-rand.$Encrypt(pk, p)$ *first computes* $r = \mathfrak{S}.PTSample(pk, p)$*, and then* $c = \mathfrak{S}.Encrypt(pk, r)$*. If a tuple of the form* $(pk, c, \cdot)$ *is already stored in* $\mathfrak{S}$-rand*'s internal state, then* $\perp$ *is returned (the operation fails); otherwise,* $\mathfrak{S}$-rand *stores the tuple* (pk, c, p)*, and returns* c *as the ciphertext.*
- $\mathfrak{S}$-rand.$Decrypt(pk, sk, c)$ *looks for a tuple of the form* (pk, c, x) *for some* x*. If such a tuple exists, then* x *is returned as decrypted plaintext; otherwise,* $p = \mathfrak{S}.Decrypt(pk, sk, c)$ *is called and* p *is returned.*

Note that the Encrypt function can fail if a duplicate ciphertext is produced, but since these ciphertexts are encryptions of random plaintexts then both the randomly chosen plaintext and randomness used in the encryption must be a repeat of a previous encryption, which happens with negligible probability. It is important not to confuse the above definition with an oracle that the adversary interacts with — the plaintext randomization of $\mathfrak{S}$ replaces calls to a PKE scheme that happen internal to a security game, and the adversary does not have direct access to these functions. Replacing a PKE scheme with its plaintext randomization typically results in a game that is much simpler to analyze, but is a good approximation to the original game. In particular, the following lemma shows that if the advantage against the plaintext-randomized game is negligible, then the advantage against the original game is also negligible.

Lemma 1 (Plaintext Randomization Lemma). *Let G be a game that makes sk-oblivious use of a plaintext-samplable public key encryption scheme $\mathfrak{S}$, and let $\mathfrak{S}$-rand be the plaintext randomization of $\mathfrak{S}$. Then, for any probabilistic adversary A that plays $\mathsf{G}^{\mathfrak{S}}$ so that the total game-playing time of A is bounded by t, the number of calls to $\mathfrak{S}.KeyGen$ is bounded by n, and the number of encryption and decryption requests for any individual key is bounded by q_e and q_d, respectively,*

$$\left|Adv_{A,\mathsf{G}^{\mathfrak{S}}} - Adv_{A,\mathsf{G}^{\mathfrak{S}\text{-rand}}}\right| \leq 2\, Adv_{PK\text{-}MU^{\mathfrak{S}}_n}(t', q_e, q_d), \text{where } t' = t + O(\log(q_e n))$$

Proof. Given adversary A that plays the game G we will construct an adversary A' that plays $\mathsf{PK\text{-}MU}_n^{\mathfrak{S}}$, so A' converts A into an adversary that attacks the basic multi-user CCA security of the underlying PKE $\mathfrak{S}$. A' starts by calling $\mathsf{PK\text{-}MU}_n^{\mathfrak{S}}.\mathsf{Initialize}(\lambda)$ and saves the list of public keys $pk_1, \cdots, pk_n$ for later use, setting $m = 0$ to track the number of public keys that are "in use" by G. A' next simulates the original adversary A and the game oracle G, replacing G's use of PKE $\mathfrak{S}$ with a specially constructed stateful PKE scheme $\widetilde{\mathfrak{S}}$ that has access to the public keys $pk_1, \cdots, pk_n$ and counter m, as well as the $\mathsf{PK\text{-}MU}_n^{\mathfrak{S}}$ oracle. $\widetilde{\mathfrak{S}}$ provides the standard three PKE functions as follows:

- $\widetilde{\mathfrak{S}}.KeyGen(1^\lambda)$: If $m = n$ (i.e., we have already used n keypairs), this operation returns $\perp$ and fails. Otherwise, $\widetilde{\mathfrak{S}}$ increments m (the number of keys in use) and returns (pk_m, pk_m) as the generated "keypair." Note that the "secret key" is really just the public key, but this is not important since we can perform all the operations below with just this information, and G's use of $\mathfrak{S}$ is sk-oblivious.
- $\widetilde{\mathfrak{S}}.Encrypt(pk_i, p)$ for valid public key pk_i: $\widetilde{\mathfrak{S}}$ computes random plaintext $r = \mathfrak{S}.PTSample(pk_i, p)$, and then computes $c = \mathsf{PK\text{-}MU}_n^{\mathfrak{S}}.\mathsf{PEncrypt}(i, p, r)$. The tuple (pk_i, c, p) is saved in $\widetilde{\mathfrak{S}}$'s state, and c is returned.
- $\widetilde{\mathfrak{S}}.Decrypt(pk_i, pk_i, c)$: Note that the pk_i here is used as both the public and the secret key, despite the notation. The decrypt function first checks to see if $\widetilde{\mathfrak{S}}$'s state contains a tuple (pk_i, c, p) for some p, and if such a tuple is found p is returned as the plaintext. Otherwise, $p = \mathsf{PK\text{-}MU}_n^{\mathfrak{S}}.\mathsf{Decrypt}(i, c)$ is returned.

Note that all calls to $\mathsf{PK\text{-}MU}_n^{\mathfrak{S}}.\mathsf{PEncrypt}$ store a tuple that includes the returned ciphertext, and the $\widetilde{\mathfrak{S}}.Decrypt$ function never calls the $\mathsf{PK\text{-}MU}_n^{\mathfrak{S}}.\mathsf{Decrypt}$ oracle with such a ciphertext, so all $Decrypt$ calls will succeed.

A' continues simulating A and $\mathsf{G}^{\widetilde{\mathfrak{S}}}$ until A outputs its final result, a, for game G , at which point A' calls $\mathsf{G}.\mathsf{IsWinner}(a)$ to check if A's output wins game $\mathsf{G}^{\widetilde{\mathfrak{S}}}$. Based on this, A' outputs its guess b' for $\mathsf{PK\text{-}MU}^{\mathfrak{S}}$'s secret bit b as follows:

- If A wins $\mathsf{G}^{\widetilde{\mathfrak{S}}}$, A' outputs guess $b' = 0$.
- If A loses $\mathsf{G}^{\widetilde{\mathfrak{S}}}$, A' outputs guess $b' = 1$.

Thus, A' wins if A wins and $b = 0$, or if A loses and $b = 1$. Since b is uniformly distributed,

$$P(A' \text{ wins}) = 0.5\, P(A \text{ wins } \mathsf{G}^{\widetilde{\mathfrak{S}}} \mid b = 0) + 0.5\, P(A \text{ loses } \mathsf{G}^{\widetilde{\mathfrak{S}}} \mid b = 1)\ . \qquad (1)$$

While our simulator does not know the bit b, the construction of $\widetilde{\mathfrak{S}}$ is such that when $b = 0$ the game played by A is exactly $\mathsf{G}^{\mathfrak{S}}$ (since in this case the encryption request is answered by an encryption of the real plaintext), and when $b = 1$ the game played by A is exactly $\mathsf{G}^{\mathfrak{S}\text{-}\mathsf{rand}}$ (since in this case the encryption request is answered using plaintext randomization). Using this observation we

can simplify (1) as follows:

$$\begin{aligned} P(A' \text{ wins}) &= 0.5\, P(A \text{ wins } \mathsf{G}^{\mathfrak{S}}) + 0.5\, P(A \text{ loses } \mathsf{G}^{\mathfrak{S}\text{-rand}}) \\ &= 0.5\, P(A \text{ wins } \mathsf{G}^{\mathfrak{S}}) + 0.5\, \left(1 - P(A \text{ wins } \mathsf{G}^{\mathfrak{S}\text{-rand}})\right) \\ &= 0.5\, + 0.5\, P(A \text{ wins } \mathsf{G}^{\mathfrak{S}}) - 0.5\, P(A \text{ wins } \mathsf{G}^{\mathfrak{S}\text{-rand}})\ . \end{aligned}$$

By definition, $Adv_{A',\text{PK-MU}_n^{\mathfrak{S}}} = |P(A' \text{ wins }) - 0.5|$, and since A' only does some simple table lookups in addition to its simulation of A and G, which induces at most q_e and q_d encryption and decryption requests, respectively, we can bound $Adv_{A',\text{PK-MU}_n^{\mathfrak{S}}}$ by $Adv_{\text{PK-MU}_n^{\mathfrak{S}}}(t', q_e, q_d)$, where $t' = t + O(\log(nq_e))$. It follows that:

$$\left|0.5\, P(A \text{ wins } \mathsf{G}^{\mathfrak{S}}) - 0.5\, P(A \text{ wins } \mathsf{G}^{\mathfrak{S}\text{-rand}})\right| \leq Adv_{\text{PK-MU}_n^{\mathfrak{S}}}(t', q_e, q_d)\ ,$$

and so,

$$\left|P(A \text{ wins } \mathsf{G}^{\mathfrak{S}}) - P(A \text{ wins } \mathsf{G}^{\mathfrak{S}\text{-rand}})\right| \leq 2\ Adv_{\text{PK-MU}_n^{\mathfrak{S}}}(t', q_e, q_d)\ . \qquad (2)$$

Returning to the original problem, comparing the advantage of A with respect to $\mathsf{G}^{\mathfrak{S}}$ and $\mathsf{G}^{\mathfrak{S}\text{-rand}}$,

$$\left|Adv_{A,\mathsf{G}^{\mathfrak{S}}} - Adv_{A,\mathsf{G}^{\mathfrak{S}\text{-rand}}}\right| = \left|\ |P(A \text{ wins } \mathsf{G}^{\mathfrak{S}}) - 0.5| - |P(A \text{ wins } \mathsf{G}^{\mathfrak{S}\text{-rand}}) - 0.5|\ \right|\ ,$$

and so,

$$\left|Adv_{A,\mathsf{G}^{\mathfrak{S}}} - Adv_{A,\mathsf{G}^{\mathfrak{S}\text{-rand}}}\right| \leq \left|\ P(A \text{ wins } \mathsf{G}^{\mathfrak{S}}) - P(A \text{ wins } \mathsf{G}^{\mathfrak{S}\text{-rand}})\ \right| \qquad (3)$$

Combining bounds (2) and (3), we conclude that

$$\left|Adv_{A,\mathsf{G}^{\mathfrak{S}}} - Adv_{A,\mathsf{G}^{\mathfrak{S}\text{-rand}}}\right| \leq 2\ Adv_{\text{PK-MU}_n^{\mathfrak{S}}}(t', q_e, q_d),$$

which is the bound claimed in the lemma statement. □

4 Hybrid Encryption

In this section we provide an example of how plaintext randomization can simplify proofs other PKE-hybrid protocols, by giving a very simple proof for the security of hybrid encryption using a standard PKE rather than a KEM.

Definition 7. *A hybrid encryption scheme $\mathcal{H}(\mathcal{P}, \mathcal{S})$ combines the use of a PKE scheme $\mathcal{P}$ and an SKE scheme $\mathcal{S}$ to produce a public-key encryption scheme defined by the following functions:*

- *$KeyGen(1^\lambda)$: Compute $(pk, sk) = \mathcal{P}.KeyGen(1^\lambda)$ and return the keypair (pk, sk).*
- *$Encrypt(pk, p)$: Compute $k = \mathcal{S}.KeyGen(1^\lambda)$ and then ciphertexts $\psi = \mathcal{P}.Encrypt(pk, k)$ and $\phi = \mathcal{S}.Encrypt(k, p)$. The returned ciphertext is the pair $c = (\psi, \phi)$.*

– *Decrypt*(pk, sk, c)*: Ciphertext* $c = (\psi, \phi)$ *is decrypted by first finding the session key using* $k = \mathcal{P}.Decrypt(pk, sk, \psi)$ *and then computing the final plaintext* $p = \mathcal{S}.Decrypt(k, \phi)$.

Since the net effect of a hybrid encryption scheme is the same as a public key encryption system, security is defined by the multi-user CCA security notion that was given in Section 2.1. We also refer to the multi-user CCA security of a symmetric encryption scheme in the following analysis. The SK-MU game is a straightforward modification of PK-MU to a symmetric encryption setting, and is omitted here.

Theorem 1. *For any hybrid encryption scheme* $\mathcal{H}(\mathcal{P}, \mathcal{S})$*, if* A *is an adversary that runs in time* t *in a game that uses at most* n *keypairs and performs at most* q_e *and* q_d *encryption and decryption queries, respectively, then*

$$Adv_{A,PK\text{-}MU_n^{\mathcal{H}}} \leq 2\ Adv_{PK\text{-}MU_n^{\mathcal{P}}}(t', q_e, q_d) + Adv_{SK\text{-}MU_{q_e n}^{\mathcal{S}}}(t', 1, q_d)\ ,$$

where $t' = t + O(\log(q_e n))$.

Proof. Consider the plaintext randomization of $\mathcal{H}$, and an adversary A playing the $\mathsf{PK\text{-}MU}_n^{\mathcal{H}\text{-}\mathsf{rand}}$ game: in this situation, a ciphertext (ψ, ϕ) is such that ψ is completely unrelated to ϕ — it is just an encryption of a random value. Therefore, any attack on $\mathcal{H}$-rand can be immediately turned into an attack on the SKE scheme $\mathcal{S}$ by simulating A and adding encryptions of random values to simulate the ψ portion of each ciphertext. Each use of the SKE scheme in this scenario uses a different random key, with a single encryption per symmetric key, and since the probability of winning the SKE game is the same as A winning the $\mathcal{H}$-rand game, we can bound the advantage of A in the $\mathsf{PK\text{-}MU}_n^{\mathcal{H}\text{-}\mathsf{rand}}$ game as

$$Adv_{A,\mathsf{PK\text{-}MU}^{\mathcal{H}\text{-}\mathsf{rand}}} \leq Adv_{\mathrm{SK\text{-}MU}_{q_e n}^{\mathcal{S}}}(t, 1, q_d)\ .$$

By the Plaintext Randomization Lemma, we know that

$$\left|Adv_{A,\mathsf{PK\text{-}MU}^{\mathcal{H}}} - Adv_{A,\mathsf{PK\text{-}MU}^{\mathcal{H}\text{-}\mathsf{rand}}}\right| \leq 2Adv_{\mathsf{PK\text{-}MU}^{\mathcal{P}}}(t', q_e, q_d),$$

leading to the bound in the theorem. □

While we generally state bounds in terms of concrete security, we restate this bound as a simple statement about security with respect to a probabilistic polynomial-time (PPT) adversary. In particular, we say simply that a scheme is "PK-CCA2-secure" if $Adv_{A,\mathsf{PK\text{-}CCA2}}$ is negligible for any PPT adversary A, and similarly for other games.

Corollary 1. *If* $\mathcal{P}$ *is a PK-CCA2-secure PKE scheme, and* $\mathcal{S}$ *is a SK-CCA2-secure SKE scheme, then hybrid scheme* $\mathcal{H}(\mathcal{P}, \mathcal{S})$ *is a PK-CCA2-secure PKE scheme.*

5 Secret Sharing with Encrypted Shares

Loosely speaking, a secret sharing scheme consists of a dealer, who knows some secret, and a set of participants who are given shares of this secret by the dealer in such a way that only authorized sets of participants can reconstruct the secret from their shares. Formalizing this intuitive notion is not difficult, and the formulation we use is a slight modification of the 1987 definition given by Ito, Saito, and Nishizeki [10].

Formally, an n-way secret sharing scheme is one in which a secret s comes from some space of secrets $\mathcal{SS}$, and a function MakeShares: $\mathcal{SS} \rightarrow \mathcal{PS}^n$ where $\mathcal{PS}$ is the "piece space" for shares of the secret. In particular, given $s \in \mathcal{SS}$, MakeShares(s) produces a vector $(s_1, s_2, \cdots, s_n)$ of shares, where $s_i \in \mathcal{PS}$ for all i, and shares are identified by their index in this vector. Using notation $[n] = \{1, \cdots, n\}$, an *access structure* $\Gamma \subseteq 2^{[n]}$ is a set of subsets of indices that are authorized to reconstruct the secret. For any subset $(i_1, i_2, \cdots, i_m) \in \Gamma$ we can use function Reconstruct$(s_{i_1}, s_{i_2}, \cdots, s_{i_m}) = s$ to reconstruct s from shares identified by an authorized subset of indices. We require that if $(i_1, i_2, \cdots, i_m) \notin \Gamma$ then it is infeasible to reconstruct s from shares $s_{i_1}, s_{i_2}, \cdots, s_{i_m}$, where "infeasible" can be defined either from a computational or an information theoretic standpoint. All of the secret sharing schemes we consider in this paper use the information theoretic model of security, meaning that for any $(i_1, i_2, \cdots, i_m) \notin \Gamma$ all secrets $s \in \mathcal{SS}$ are equally likely, so no algorithm can extract any information about s from these shares — such a secret sharing scheme is called *perfect*. Perfect secret sharing schemes are known and widely used for certain specific access structures, such as Shamir's secret sharing scheme for threshold access structures [11]. As is typical in secret sharing work, we require that Γ be monotone, so that if some $\mathcal{I} \in \Gamma$ then all supersets of $\mathcal{I}$ are also in Γ.

We are interested in schemes in which shares are encrypted using a public key scheme, so for a PKE with plaintext space $\mathcal{PT}$ we assume an embedding function $e : \mathcal{PS} \rightarrow \mathcal{PT}$ and a "de-embedding" function $d : \mathcal{PT} \rightarrow \mathcal{PS}$ so that $d(e(s_i)) = s_i$ for any share s_i. In what follows, use of the embedding and de-embedding functions is implicitly assumed when shares are encrypted or decrypted. Despite the widespread use of secret sharing, both as an independent cryptographic functionality and as a component of other cryptographic protocols, we have not located any formal work analyzing the situation in which shares are encrypted using a public key encryption scheme. Such a situation is similar to hybrid encryption, but with the ultimate secret computed using a combination of decryption and secret reconstruction. Traditional notions of indistinguishability games have serious challenges in this setting. Traditional games typically have the restriction that once a ciphertext is produced by a challenge oracle, the adversary is not allowed to request and receive a decryption of that ciphertext, so any reduction from a standard indistinguishability game should similarly avoid queries on ciphertexts produced by the challenge oracle. In the setting of secret sharing, we want to allow decryption of challenge ciphertexts as long as the full set of decryptions produced does not correspond to a set $\gamma \in \Gamma$ that can reconstruct the secret. Our plaintext randomization technique can handle this situation quite

cleanly, while it is not clear how to start with a reduction from a traditional ciphertext indistinguishability game.

In our formalization, we consider an n-way perfect secret sharing scheme as defined above with monotone access structure Γ for which we will use k different keypairs (numbered 1 to k for convenience) to encrypt these shares according to a key mapping function $\mathcal{K} : [n] \to [k]$ so that share i will be encrypted using key number $\mathcal{K}(i)$. The scheme may depend on the access structure, and may in fact only work for some specific subset of access structures (such as threshold access structures), and we assume there is some appropriate and compact description $\mathcal{D}(\Gamma)$ of valid Γ's for a particular scheme.

Using this general key mapping function allows us to capture several different scenarios within one general definition. For example, with $k = 1$ and $\mathcal{K}(i) = 1$ for all i, we capture a scheme in which all shares are encrypted with the key of a single trusted authority and shares are distributed (either to a single party or multiple parties) so that decryption of shares is allowed based on some criteria determined by the trusted authority. For example, Γ might be a threshold structure with threshold t, and trusted hardware controls how many shares may be decrypted at any particular time (this is the case in Gunupudi and Tate's generalized oblivious transfer scheme based on trusted hardware [8]). As another example, consider $k = n$ escrow authorities, all with their own keys, and $\mathcal{K}(i) = i$ for all i. In this case, a single user could obtain all encrypted shares, and if the user needs to open the secret at some point the user could approach some subset $\gamma \in \Gamma$ of escrow authorities and convince them that they should decrypt the share that is encrypted with their key. This is the scenario described in the Introduction.

We define the following security game for secret sharing with encrypted shares.

- $\mathsf{ESS}^{\mathfrak{S}}.\mathsf{Initialize}(1^\lambda, k, n, \mathcal{K}, \mathcal{D}(\Gamma))$: If this scheme does not support access structure Γ, return $\perp$ and do not answer any further queries. Otherwise, the oracle generates keypairs $(pk_i, sk_i) = \mathfrak{S}.KeyGen(1^\lambda)$ for $i = 1, \ldots, k$, picks a random bit $b \in \{0, 1\}$, and saves n, $\mathcal{K}$, and $\mathcal{D}(\Gamma)$ for later use. γ, used to track the set of shares that have been decrypted, is initialized to the empty set. Finally, set flag c to false, indicating that no challenge has yet been made, and return $pk_1, \ldots, pk_k$.
- $\mathsf{ESS}^{\mathfrak{S}}.\mathsf{Challenge}(x_0, x_1)$: If c is true, then return $\perp$. Otherwise, the oracle generates n shares $s_1, \ldots, s_n$ for x_b, creates share ciphertexts by computing $c_i = \mathfrak{S}.Encrypt(pk_{\mathcal{K}(i)}, s_i)$ for $i = 1, \ldots, n$, sets flag c to true, and returns the vector $(c_1, \ldots, c_n)$ to the adversary (keeping a copy for future reference).
- $\mathsf{ESS}^{\mathfrak{S}}.\mathsf{Decrypt}(i, x)$ (where $i \in [k]$ is a key number and x is a ciphertext): If $x \notin \{c_j \mid \mathcal{K}(j) = i\}$ (so c_j was not produced by encrypting with key i), then simply compute and return $\mathfrak{S}.Decrypt(pk_i, sk_i, x)$. Otherwise, $x = c_j$ for some j with $\mathcal{K}(j) = i$, so test whether $\gamma \cup \{j\} \in \Gamma$: if it is, return $\perp$; otherwise, set $\gamma \leftarrow \gamma \cup \{j\}$ and return $\mathfrak{S}.Decrypt(pk_i, sk_i, x)$.
- $\mathsf{ESS}^{\mathfrak{S}}.\mathsf{IsWinner}(a)$: Takes a bit a from the adversary, and returns true if and only if $a = b$.

Note that the test in $\mathsf{ESS}^{\mathfrak{S}}.\mathsf{Decrypt}(i, x)$ disallows decrypting shares for a share set that would allow reconstruction of the secret, while still allowing some decryptions. Furthermore, since Γ is monotone, if the ultimate set of shares that have been decrypted are from a set $\gamma \notin \Gamma$, then every subset of γ is also not in Γ and so this test does not restrict adversaries in any way other than not allowing them to receive a full share set.

Since the bounds in the Plaintext Randomization Lemma depend on the number of times each key is used, we define q_e to be the largest number of times any key is used according to our key mapping function. Specifically,

$$q_e = \max_{i \in [k]} |\{j \,|\, \mathcal{K}(j) = i\}| \ .$$

Theorem 2. *If ESS is a k-key perfect n-way secret sharing scheme using PKE $\mathfrak{S}$, with q_e and q_d defined as described above, then for any time t adversary A,*

$$Adv_{A,\mathit{ESS}^{\mathfrak{S}}} \leq 2\, Adv_{\mathit{PK\text{-}MU}_n^{\mathfrak{S}}}(t', q_e, q_d).$$

Proof. Consider the $\mathsf{ESS}^{\mathfrak{S}\text{-}\mathsf{rand}}$, the ESS game using the plaintext randomized version of $\mathfrak{S}$. Let b denote the secret random bit chosen during the initialization step, let x_0 and x_1 denote the challenge secrets selected by A, and let $c_1, c_2, \cdots, c_n$ be the ciphertexts produced by the ESS oracle encrypting shares of x_b using $\mathfrak{S}$-rand. At any time while A is playing $\mathsf{ESS}^{\mathfrak{S}\text{-}\mathsf{rand}}$, its view consists of ciphertexts $c_1, c_2, \cdots, c_n$ as well as a set of shares that it has decrypted, $s_{i_1}, s_{i_2}, \cdots, s_{i_m}$, where $\{i_1, i_2, \cdots, i_m\} = \gamma \notin \Gamma$ (as enforced by the ESS game). Since the decrypted shares are not in Γ and the secret sharing scheme is perfect, and since plaintext randomization results in ciphertexts being independent of the actual shares (and hence the secret), each secret (x_0 or x_1) is equally likely given the adversary's view. Therefore, $Adv_{A,\mathsf{ESS}^{\mathfrak{S}\text{-}\mathsf{rand}}} = 0$.

Referring back to the Plaintext Randomization Lemma,

$$\left| Adv_{A,\mathsf{ESS}^{\mathfrak{S}}} - Adv_{A,\mathsf{ESS}^{\mathfrak{S}\text{-}\mathsf{rand}}} \right| \leq 2\, Adv_{\mathrm{PK\text{-}MU}_n^{\mathfrak{S}}}(t', q_e, q_d),$$

and since $Adv_{A,\mathsf{ESS}^{\mathfrak{S}\text{-}\mathsf{rand}}} = 0$ we conclude with the bound stated in the theorem. □

6 Conclusions

In this paper we have examined the problem of encrypted secret sharing (ESS), developing a general and flexible definition, and solving challenges posed in the analysis by developing a new, general-purpose, and powerful technique called plaintext randomization. This technique modularizes the analysis PKE-hybrid cryptographic protocols, and separates out the dependence on the security of the PKE scheme in a way that is captured by the Plaintext Randomization Lemma that is proved in this paper. Beyond the immediate result for ESS, we believe that plaintext randomization holds great promise for additional applicability, given the prevalence of public-key encryption in cryptographic schemes and the efficient schemes that can be derived in the reduction-based security model.

References

1. Bellare, M., Boldyreva, A., Micali, S.: Public-key encryption in a multi-user setting: Security proofs and improvements. In: Proc. of EUROCRYPT. (2000) 259–274
2. Bellare, M., Desai, A., Jokipii, E., Rogaway, P.: A concrete security treatment of symmetric encryption. In: Proceedings of the 38th Symposium on Foundations of Computer Science (FOCS). (1997) 394–403
3. Cramer, R., Shoup, V.: Design and analysis of practical public-key encryption schemes secure against adaptive chosen ciphertext attack. SIAM J Comput. **33** (2003) 167–226
4. Dwork, C., Naor, M., Reingold, O., Stockmeyer, L.: Magic functions: In memoriam: Bernard M. Dwork 1923–1998. Journal of the ACM **50**(6) (November 2003) 852–921
5. Goldreich, O., Krawczyk, H.: On the composition of zero-knowledge proof systems. SIAM Journal on Computing **25**(1) (1996) 169–192
6. Goldreich, O., Micali, S., Wigderson, A.: How to play any mental game. In: Proc. of the 19th Symposium on Theory of Computing (STOC). (1987) 218–229
7. Goldwasser, S., Micali, S.: Probabilistic encryption. Journal of Computer and System Sciences **28**(2) (1984) 270–299
8. Gunupudi, V., Tate, S.R.: Generalized non-interactive oblivious transfer using count-limited objects with applications to secure mobile agents. In: Proc. of Financial Cryptography. Volume 5143 of Springer Lecture Notes in Computer Science. (2008) 98–112
9. Hofheinz, D., Jager, T.: Tightly secure signatures and public-key encryption. In: Proc. of CRYPTO. (2012) 590–607
10. Ito, M., Saito, A., Nishizeki, T.: Secret sharing scheme realizing general access structure. In: Proc. of IEEE Globecom. (1987) 99–102
11. Shamir, A.: How to share a secret. Communications of the ACM **22(11)** (1979) 612–613

A Single-User CCA Security and Results

Up until this point, we have stated results in their most general form, using the multiple-key version of CCA PKE security defined in Section 2.1. In many settings, we are only interested in a single key version of security, and consequently can use simpler forms of the results. In this section we present results specific to the classical single-key notion of CCA security.

A.1 Chosen Ciphertext Security Definition

In this section, we define the adaptive chosen ciphertext security game (the standard "CCA2" game) that is widely used for describing security of public-key encryption schemes. The basic idea is that the adversary gets to pick two plaintexts for a "challenge," the oracle encrypts one randomly-chosen plaintext and returns the corresponding ciphertext to the adversary. The adversary can request as many ciphertext decryptions as it likes, either before or after the challenge is submitted (which is what makes this the adaptive, or CCA2, version),

with the only restriction being that the oracle cannot be asked to decrypt the ciphertext returned from the challenge. These conditions are all reflected in the definition below.

- $\mathsf{PK\text{-}CCA2}^{\mathfrak{S}}.\mathsf{Initialize}(\lambda)$: The oracle generates a keypair $(pk, sk) = \mathfrak{S}.KeyGen(1^{\lambda})$, picks a random bit $b \in \{0, 1\}$, sets cc (for "challenge ciphertext") to null, and returns pk.
- $\mathsf{PK\text{-}CCA2}^{\mathfrak{S}}.\mathsf{Decrypt}(x)$: If $x = cc$, the oracle returns $\perp$ to the adversary; otherwise, it computes and returns $\mathfrak{S}.Decrypt(pk, sk, x)$.
- $\mathsf{PK\text{-}CCA2}^{\mathfrak{S}}.\mathsf{Challenge}(x_0, x_1)$: If cc is non-null (i.e., Challenge has been called previously), the oracle returns $\perp$; otherwise, it sets $cc = \mathfrak{S}.Encrypt(pk, x_b)$ and returns cc to the adversary.
- $\mathsf{PK\text{-}CCA2}^{\mathfrak{S}}.\mathsf{IsWinner}(a)$: Takes a bit a from the adversary, and returns true if and only if $a = b$.

To be "CCA2-secure," a PKE $\mathfrak{S}$ should be such that it is impossible for a probabilistic polynomial time adversary to win this game, i.e., the advantage of any adversary in this game is negligible. This is a widely-used notion of security for PKE schemes — while weaker notions of PKE security are possible (e.g., chosen plaintext, or non-adaptive chosen ciphertext), there are efficient schemes that have CCA2 security under reasonable complexity assumptions. For example, the Cramer-Shoup public-key encryption scheme is CCA2-secure in the plain model, under the decisional Diffie-Hellman assumption [3].

A.2 Plaintext Randomization Lemma

We next provide a corollary that gives two alternate forms for the Plaintext Randomization Lemma, which might be useful if a bound is desired in terms of traditional single-user, single-challenge CCA2 security. This corollary is stated without proof, as the bounds follow directly from the Plaintext Randomization Lemma and multi-user PKE bounds proved by Bellare *et al.* [1]. The second bound follows from the tighter multi-user security that is possible from the Cramer-Shoup PKE scheme.

Corollary 2. *Let G be a game that makes sk-oblivious use of a plaintext-samplable public key encryption scheme $\mathfrak{S}$, and let $\mathfrak{S}$-rand be the plaintext randomization of $\mathfrak{S}$. Then, for any probabilistic adversary A that plays $\mathsf{G}^{\mathfrak{S}}$ so that the total game-playing time of A is bounded by t, the number of calls to $\mathfrak{S}.KeyGen$ is bounded by n, and the number of encryption and decryption requests for any individual key is bounded by q_e and q_d, respectively,*

$$\left|Adv_{A,\mathsf{G}^{\mathfrak{S}}} - Adv_{A,\mathsf{G}^{\mathfrak{S}\text{-}\mathsf{rand}}}\right| \leq 2q_e n\, Adv_{PK\text{-}CCA2^{\mathfrak{S}}}(t', q_d) \ ,$$

where $t' = t + O(\log(q_e n))$. Furthermore, for the Cramer-Shoup PKE scheme, denoted $\mathcal{CS}$, we can bound

$$\left|Adv_{A,\mathsf{G}^{\mathcal{CS}}} - Adv_{A,\mathsf{G}^{\mathcal{CS}-\mathsf{rand}}}\right| \leq 2q_e\, Adv_{PK\text{-}CCA2^{\mathcal{CS}}}(t', q_d) \ .$$

A.3 Encrypted Secret Sharing in the Single-Key Model

Restricting the encrypted secret sharing (ESS) problem so that only a single key is used (and so no key mapping function is necessary), we arrive at the following corollary, which also reflects the savings that can be achieved by using the Cramer-Shoup encryption scheme.

Corollary 3. *If* ESS *is a k-key perfect n-way secret sharing scheme using PKE $\mathfrak{S}$, with q_e and q_d defined based on key mapping function $\mathcal{K}$ as described above, then for any time t adversary A,*

$$Adv_{A,\mathsf{ESS}^{\mathfrak{S}}} \leq 2kq_e \, Adv_{PK\text{-}CCA2^{\mathfrak{S}}}(t', q_d) \ .$$

For Cramer-Shoup encryption scheme $\mathcal{CS}$, we can bound

$$Adv_{A,\mathsf{ESS}^{\mathcal{CS}}} \leq 2q_e \, Adv_{PK\text{-}CCA2^{\mathcal{CS}}}(t', q_d) \ .$$

All of the results above give concrete security bounds, so we point out the obvious high-level conclusion: if we use a perfect secret sharing scheme with a CCA2 secure encryption scheme, then no probabilistic polynomial time adversary can win the ESS game with more than negligible probability.