New Techniques for Public Key Encryption with Sender Recovery

Murali Godi

A Thesis

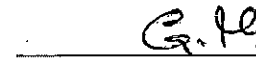
Submitted to the
Graduate Faculty of the
State University of New York Polytechnic Institute
in Partial Fulfillment of the
Requirements for the
Degree of

Master of Science

Utica, New York
December 15, 2016

New Techniques for Public Key Encryption with Sender Recovery

Murali Godi



Signature of Author

12/30/2016

Date of Graduation

NEW TECHNIQUES FOR PUBLIC KEY ENCRYPTION WITH SENDER RECOVERY

Except where reference is made to the work of others, the work described in this thesis is my own or was done in collaboration with my advisory committee. Further, the content of this thesis is truthful in regards to my own work and the portrayal of others' work. This thesis does not include proprietary or classified information.

Murali Godi

Certificate of Approval:

Roopa Viswanathan
Assistant Professor
Department of Computer and Information Sciences

Jorge Novillo
Professor
Department of Computer and Information Sciences

Chen-Fu Chiang
Assistant Professor
Department of Computer and Information Sciences

THESIS ABSTRACT

NEW TECHNIQUES FOR PUBLIC KEY ENCRYPTION WITH SENDER RECOVERY

Murali Godi

Master of Science, December 15, 2016
(B.Tech., JNTUH, 2014)

37 Typed Pages

Directed by Roopa Viswanathan

In this paper, we consider a situation where a sender transmits a ciphertext to a receiver using a public-key encryption scheme, and at a later point of time, wants to retrieve the plaintext, without having to request the receiver's help in decrypting the ciphertext, and without having to store a set of plaintext/ciphertext pairs for every receiver the sender interacts with. This problem, known as public key encryption with sender recovery has intuitive solutions based on KEM/DEM schemes. We propose a KEM/DEM-based solution that is CCA-secure, and only requires the receiver to be equipped with a public/secret key pair (the sender needs only a symmetric recovery key), and has much simplified proofs compared to prior work in this area. We prove our protocols secure in the single receiver and multi-receiver setting. To achieve our goals, we use an analysis technique called plaintext randomization that results in greatly simplified and intuitive proofs for protocols that use a PKE internally as a component and compose the PKE with other primitives. We instantiate our protocol for public key encryption with sender recovery with the well-known KEM/DEM scheme due to Cramer and Shoup.

Style manual or journal used Journal of Approximation Theory (together with the style known as "sunypolyms"). Bibliography follows van Leunen's *A Handbook for Scholars*.

Computer software used The document preparation package TEX (specifically LATEX2e) together with the style-file `sunypolyms.sty`.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

1.1 Motivation

Consider a situation where Alice and Bob exchange e-mails through an *untrusted encrypted* e-mail service provider. Alice sends e-mails to Bob encrypted under his public key, and does not necessarily save a plaintext copy of every e-mail she sends Bob on her local device. Additionally, Alice uses this encrypted email service to send large volumes of e-mails to multiple recipients, hence locally saving either a plaintext or encrypted copy of every email she sends out may not be a feasible option for her. She also does not want to save a copy of every plaintext e-mail on the untrusted server, since she does not want to trust the party or parties running the server. In this scenario, Alice cannot retrieve a plaintext message at a later date without the co-operation of the receiver of the message, who is presumably the only party who will have the corresponding Private key used to decrypt the message. Ideally, Alice should be able to retrieve the plaintext messages without having to contact Bob (or other recipients), who either may not be available, or may not have any incentive to co-operate with Alice at a later date. Alice could possibly create two copies of the encrypted email, one under Bob's public key, and one under her own public key, and store the second copy on the untrusted server. Besides the redundancy of requiring every message to be encrypted twice, this solution consumes valuable space on the server, who might have quotas for each user. In a typical situation where a single user sends hundreds of emails in a day, storing a second copy of every email under her own key isn't really a feasible solution. Another solution would be to use the services of a trusted third party, Trent under whose public key Alice will encrypt a copy of the message, and possible store the copy on Trent's side. But this still would require Alice to trust Trent, and requires every message to be encrypted twice. This problem naturally does not exist if Alice and Bob had used symmetric key cryptography, but we believe this problem is still

worth looking into, as public key cryptography is widespread, might be used in legacy applications, and users may not always get to pick the choice of encryption scheme, especially when using the services of third party email service providers.

This problem arises in other contexts too, such as a "free-mium" online collaborative file sharing service, e.g., Dropbox or SpiderOak, that allow users to store encrypted data up to a certain limit (e.g., Dropbox free limit is 2GB), and charge users for anything above that limit. Consider a file sharing service that allows users to store files encrypted under their respective public keys [1]. Such a service provides users the option of creating a shared folder for a group, but instead of allowing all group members to view all files, a user can encrypt files under the public keys of select group members, such that only that subset of group members can view the files. Any data that is meant for a user is put in the shared folder encrypted under that user's public key. Furthermore, users are not inclined to store a second copy of all data that they exchange, encrypted under their own keys, since that would quickly use up the group's quota. All users can either use a common PKE scheme, or the service provider can give each user the option of choosing from a set of PKE schemes. In this situation, a user cannot decrypt files they have encrypted under the keys of other group members, without having to store a copy locally.

This notion of a sender being able to decrypt a ciphertext encrypted under the key of a receiver, without the co-operation of the receiver, and without having to store a second copy (either plaintext, or encrypted under their own public key), known as *public key encryption with sender recovery*, was first introduced by Wei et al. [11] as a complementary notion to forward secrecy, in which past encrypted messages cannot be decrypted using valid, but old keys. In this paper, we provide new construction for the same problem using KEM/DEM schemes, with no requirement on the sender to be equipped with a public/secret key pair. Our constructions use a proof technique called plaintext randomization [10], which results in greatly simplified, intuitive, and cleaner proofs compared to prior work.

[1] currently SpiderOak is the only file sharing cloud-based service that offers users the option of storing data encrypted under their own keys, and uses hybrid encryption: 2048-bit RSA and 256-bit AES

1.2 Our Contributions

- We propose a new protocol for public-key encryption with sender recovery, where the sender can independently retrieve a plaintext message that was encrypted under the receiver's public key, without receiving help from the receiver, and without having to store a local copy, encrypted or otherwise. Potential applications of our protocol include untrusted third party data storage (e.g., Dropbox), and encrypted e-mail recovery services.

- We consider two models: the single receiver and multiple receiver model, and make minimal assumptions with respect to the keys. In particular, we only require the sender to be equipped with a single symmetric key that can be used across multiple receivers. We use key encapsulation mechanism/data encapsulation mechanism (KEM/DEM) [6] based hybrid encryption combined with a technique called *plaintext randomization*, and helps us obtain proofs of CCA2 security that are clean, intuitive, and much simpler compared to prior work in this area.

- We instantiate our protocol using the classic Cramer-Shoup KEM/DEM-based hybrid encryption scheme. We prove that our protocols are CCA2 secure if the underlying PKE scheme and DEM scheme is CCA2 secure, i.e., we do not require the underlying KEM scheme to be secure in any sense (all previous work in this area required both the KEM and DEM schemes to be CCA2 secure).

CHAPTER 2

RELATED WORK

Wei, Zheng, and Wang [13] first introduced the idea of public-key encryption with sender recovery using KEM/DEM-based hybrid encryption protocols. The idea of a sender being able to recover a previously encrypted ciphertext without the receiver's help is somewhat complementary to the better-known notion of forward security [4, 3] where the goal is to prevent decryption of ciphertexts using old, expired keys. The scheme of [13] required both, sender and receiver to be equipped with a public/secret keypair, and among other things, [13] provided the sender the ability to authenticate the ciphertext to check if it really originated from her, and worked for multiple receivers. In further work, Wei and Zheng [11, 12] presented an efficient public key encryption scheme with sender recovery, but which requires only the receiver to have a public/secret keypair. The efficiency gains though, come at the cost of sacrificing ciphertext authenticity checks, besides their scheme works only in the single receiver model. The main difference between prior work and our paper is that [13, 11, 12] rely on the underlying KEM/DEM scheme to be CCA2 secure in order to prove the security of their protocols. We considerably relax this requirement, and *do not* require the KEM/DEM scheme to be CCA2 secure. One can implement our constructions using any general-purpose public-key encryption scheme, combined with a CCA2 secure symmetric key encryption scheme. Furthermore, in our work, we consider the multiple receiver model, where a sender can recover ciphertexts encrypted under the keys of multiple receivers, using just one symmetric recovery key (no requirement for the sender to have a separate recovery key for each receiver).

More generally, KEM/DEM schemes have been used in applications such as identity-based password exchange [6], puncturable encryption [9], attribute-based encryption [5], leakage resilient cryptosystems [8]. We do not review the vast KEM/DEM literature here, since we are not designing

a new KEM/DEM scheme, rather we are using KEM/DEM to build a public-key encryption scheme that allows for sender recovery.

CHAPTER 3

PRELIMINARIES

In this section we review some relevant definitions and concepts that will be used in the construction of our protocols. We start with the basic public-key encryption with sender ecovery scheme by Wei and Zhang [11].

Definition 1. *(Basic PKE-SR scheme[1] [11])*

1. $(PK_r, SK_r, K_s) \leftarrow \mathsf{KeyGen}(1^\lambda)$*: This is a randomized algorithm that outputs a public/secret keypair for the receiver,* (PK_r, SK_r) *and a secret recovery key,* K_s *for the sender. This is run by both parties individually to generate their respective keys.*

2. $c \leftarrow \mathsf{Encrypt}(K_s, PK_r, m)$*. This is a randomized algorithm run by the sender that takes as input the sender's recovery key, the receiver's public key, a message* m*, and gives as output a ciphertext* c*. Internally, it consists of two algorithms,* KEMEncrypt *and* DEMEncrypt*. The sender picks a* $\tau \leftarrow \{0,1\}^\lambda$*, and computes* $r = F(K_s, \tau, PK_r)$*, where* F *is an injective function. The sender then generates and encrypts an ephemeral, shared, session key,* $(c_{KEM}, \kappa) \leftarrow \mathsf{KEMEncrypt}(r, PK_r)$ *. The sender then encrypts the message using the ephemeral key,* $(c_{DEM}) \leftarrow \mathsf{DEMEncrypt}(\kappa, m)$*. Set* $c = (c_{KEM}, c_{DEM})$*. Send* c *to receiver.*

3. $m \leftarrow \mathsf{Decrypt}(c, PK_r, SK_r)$*. This is a deterministic algorithm run by the receiver to extract* m *from the ciphertext* $c = (c_{KEM}, c_{DEM})$*. The receiver first retrieves the ephemeral key,* $\kappa \leftarrow \mathsf{KEMDecrypt}(SK_r, c_{KEM})$*. Then the receiver then retrieves the message,* $m \leftarrow \mathsf{DEMDecrypt}(\kappa, c_{DEM})$.

[1] While it is possible to, and would be trivial to introduce a message authentication code in the scheme for integrity checking, we omit that step here for clarity of presentation.

4. $m \leftarrow \mathsf{Recover}(K_s, PK_r, c)$*: This is a deterministic algorithm run by the sender to recover the message* m *from the ciphertext* $c = (c_{KEM}, c_{DEM})$*. The sender first computes* $r \leftarrow F(K_s, \tau, PK_r)$*, and retrieves* κ*:* $\kappa \leftarrow \mathsf{KEMRecover}(r, PK_r, C_{KEM})$*. Finally the sender recovers* $m \leftarrow \mathsf{DEMDecrypt}(K_s, c_{DEM})$

□

We next review a few definitions from[10, 1] regarding public-key encryption, plaintext-samplable public-key encryption, public-key encryption with multiple users, secret-key oblivious encryption, and plaintext randomization.

Definition 2. *(Public-Key Encryption (PKE)) 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* $Decrypt(pk, sk, Encrypt(pk, p)) = p$*.* □

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

- $PTSample : \mathcal{PK} \times \mathcal{PT} \rightarrow \mathcal{PT}$ — *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* $|x| = |p|\}$*.*

□

Definition 4. *(SK-oblivious game) 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* $\mathfrak{S}$*".* □

Definition 5. *(Public key encryption with multiple users [1]) Public key encryption in a multi-user setting is defined as follows:*

- PK-MU$_n^{\mathfrak{S}}$.*Initialize*(1^λ)*: 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.*

- PK-MU$_n^{\mathfrak{S}}$.*Decrypt*(i, x)*: If* $(i, x) \in C$*, the oracle returns* $\perp$*; otherwise, it returns* $\mathfrak{S}.Decrypt(pk_i, sk_i, x)$*.*

- PK-MU$_n^{\mathfrak{S}}$.*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.*

- PK-MU$_n^{\mathfrak{S}}$.*IsWinner*(a)*: Takes a bit* a *from the adversary, and returns* true *if and only if* $a = b$*.*

□

Definition 6. *(Plaintext randomization of a PKE) 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* $\mathfrak{S}_{\text{-rand}}$*, defined as follows:*

- $\mathfrak{S}_{\text{-rand}}.KeyGen(1^\lambda)$ *computes* $(pk, sk) = \mathfrak{S}.KeyGen(1^\lambda)$ *and returns* (pk, sk)*.*

- $\mathfrak{S}_{\text{-rand}}.Encrypt(pk, p)$ *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* $\mathfrak{S}_{\text{-rand}}$*'s internal state, then* $\perp$ *is returned (the operation fails); otherwise,* $\mathfrak{S}_{\text{-rand}}$ *stores the tuple* (pk, c, p)*, and returns* c *as the ciphertext.*

- $\mathfrak{S}_{\text{-rand}}.Decrypt(pk, sk, c)$ *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.*

□

CHAPTER 4

PLAINTEXT RANDOMIZATION AND OUR PROTOCOLS

4.1 Plaintext Randomization

The notion of plaintext randomization [10] was introduced to address issues of composability in high-level protocols, where we use one secure protocol as a component of another protocol, while retaining security inside the higher-level protocol. In particular, plaintext randomization deals with situations where a public-key encryption (PKE) scheme is used as a component in a higher-level protocol, and tries to simplify the analysis of the higher-level protocol, by abstracting out the analysis of the PKE scheme. Consider the problem of k-of-n secret sharing, where a secret is divided among n proxies or trusted third parties, such that a combination of any k of them can decrypt the secret. The analysis of this would require using a decryption oracle in the CCA2 game, where the adversary is allowed to query the decryption oracle with any ciphertext of its choice, except for the challenge ciphertext. In a standard CCA2 game, the decryption oracle will not return copies of the decrypted challenge ciphertext. Now, in a secret sharing scheme CCA2 game, the adversary will query the oracle with $m \leq k$ copies of the challenge ciphertext, and will require the oracle to return real, decrypted ciphertexts, for all of the $m \leq k$ share queries. Moreover we must make multiple encryptions, and they must be consistent. These issues could possibly be addressed by using Bellare's left-right oracle [2], but the decryption oracle must decrypt some of the shares consistently, and must disallow decryption of a set that will allow reconstruction of the secret. Unfortunately, a standard CCA2 decryption oracle will not allow consistent encryptions *and* correct decryption of some shares of the challenge ciphertext. This composability problem also arises in the context of a hybrid encryption system, where the key encapsulation mechanism (KEM) encrypts either a session key, or a random string, but a KEM is a single-use mechanism, and cannot be used in a consistent way across encryptions. In hybrid encryption, we compose the PKE and

shared key encryption (SKE) schemes in the sense that we establish that if the PKE scheme, and the SKE scheme are both secure in some sense (e.g., CCA2 secure), then the resulting hybrid encryption scheme is also secure in the same sense. Although the idea seems intuitive, the security of a hybrid encryption scheme was not established until the work of Cramer and Shoup [7].

The goal of plaintext randomization is to "cut" off the underlying PKE scheme from the rest of the protocol, with the motivation of simplifying proofs for PKE-hybrid systems. In plaintext randomization, each time the encrypt function is called on a plaintext p with some public key, PK, the CCA2 oracle replaces the real plaintext with a random string, r, such that $|r| = |p|$, encrypts r, and stores the tuple $(c = E_{PK}(r), PK, p)$. To provide consistent decryptions on a decryption request, the oracle looks up the stored tuple and returns the plaintext p, rather than the actual decryption of c, which would give r. Intuitively, if we think about using this scheme in a hybrid encryption system, we would first generate a session key k, encrypt a random string r generated by sampling the "plaintext" space (which is the key space), such that $|r| = |k|$. We would then generate $c = E_{PK_{\text{Recv}}}(r)$, where "Recv" is the recipient. Then, we would encrypt the message to be sent: $c_{DEM} = E_K(p)$. But, here, c_{KEM} and c_{DEM} are two independent strings – c_{KEM} is an encryption of a random string that bears no relation to c_{DEM}. Hence, in the analysis of the hybrid encryption scheme, we will not have to deal with composability issues.

Real-or-random security: The notion of plaintext randomization might seem similar to, but has some subtle differences with real-or-random security. In real-or-random security, the ROR oracle encrypts either the real or random plaintext, and the task of the adversary is to distinguish between the real and random encrypted shares with non-negligible probability. In the context ofa k-of-n secret sharing scheme, the ROR oracle is disallowed from decryptions of the right, challenge ciphertext shares. But it is essential to provide the adversary decrypted shares of the real ciphertexts, to prove that any subset of n less than k will not decrypt the secret correctly. It is this fundamental problem that plaintext randomization was designed to solve. In plaintext randomization, we move the entire PKE scheme inside the oracle, such that the oracle can give consistent decryptions of the challenge ciphertext, for some subset of n, which is less than k, the minimum number of decryptions necessary to correctly decrypt the challenge ciphertext.

4.2 Our Protocol

We now give a technique for constructing a public-key encryption with sender recovery (PKE-SR) scheme using plaintext randomization. Intuitively, our scheme works by encrypting a session key using the sender's recovery key and storing the encrypted key along with the ciphertext (on untrusted third party storage), such that the sender can independently retrieve the ciphertext when required, without having to contact the receiver, and while maintaining $O(n)$ storage cost, where n is the size of the ciphertext. This method was mentioned in passing by Wei and Zheng [11], but their paper did not provide details or elaborate on how such a scheme could be constructed and instantiated.

The main novel contribution of our paper is taking this intuitive idea and applying the plaintext randomization proof technique to the PKE scheme used, which enables us to obtain a simple, natural construction of a PKE-SR scheme, without having to make additional cryptographic assumptions as was done in prior work in this area, such as assuming the existence of hash function families with collision accessibility (inducing hash functions to produce collisions), and without requiring the KEM/DEM scheme to be CCA2 secure. Consequently, our protocols have simple, clean proofs, which are easy to reason about, and which do not require a complicated series of reductions between games. The basic construction of our PKE-SR scheme without plaintext randomization is given in Appendix A. We define our PKE-SR scheme with plaintext randomization for multiple receivers below, the single-receiver model is a simpler variant. Note that the sender needs to be equipped with just one recovery key, even in the presence of multiple receivers.

Definition 7. *PKE-SR using plaintext randomization with multiple receivers*

1. $((PK_1, SK_1), \cdots, (PK_n, SK_n), K_s) \leftarrow \mathsf{KeyGen}(1^\lambda)$*: This is a randomized algorithm that generates public/secret keys for the receivers and a symmetric recovery key for the sender. Parties run this algorithm individually to generate their respective keys.*

2. $c \leftarrow \mathsf{Encrypt}(PK_i, K_s, m)$*: This is a randomized algorithm run by the sender that takes as input a receiver's public key, the sender's recovery key, and a message* m*. The algorithm proceeds as follows:*

   (a) *Compute session key, κ:* $\kappa \leftarrow \mathsf{KeyGen}(PK_i, K_s, \{0,1\}^\lambda)$.

   (b) *Sample the session keyspace and produce a random string:* $\rho \leftarrow \mathsf{PTSample}(\kappa, PK_i)$, *such that* $|\rho| = |\kappa|$.

   (c) *Compute* $c_{KEM} \leftarrow \mathsf{KEMEncrypt}(PK_i, K_s, \rho)$. *If a tuple of the form* $(PK_i, c_{KEM}, \cdot)$ *already exists in the PKE's internal state, return* $\perp$. *Else store the tuple* (PK_i, c_{KEM}, κ)

   (d) *Encrypt the message* m: $c_{DEM} \leftarrow \mathsf{DEMEncrypt}(K_s, \kappa, m)$, *and set* $c = (c_{KEM}, c_{DEM})$.

   (e) *For enabling recovery at a later stage, compute* $c' = \mathsf{DEMEncrypt}(K_s, \kappa)$, *and set* $c'' = c||c'$.

   (f) *Store* c'', *and return* c.

3. $m \leftarrow \mathsf{Decrypt}(PK_i, SK_i, c)$: *This is a deterministic algorithm run by the receiver that takes as input a ciphertext* c, *the receiver's public and secret keys, and outputs the message* m. *The algorithm follows the steps below:*

   (a) *Retrieve the session key* κ: $\kappa \leftarrow \mathsf{KEMDecrypt}(PK_i, SK_i, c)$. *The PKE scheme internally looks for a tuple of the form* (PK_i, c_{KEM}, κ). *If such a tuple exists,* κ *is returned as the decrypted session key, else* $\kappa \leftarrow \mathsf{KEMDecrypt}(SK_i, c_{KEM})$ *is returned.*

   (b) *Decrypt the message* m: $m \leftarrow \mathsf{DEMDecrypt}(\kappa, c_{DEM})$.

4. $m \leftarrow \mathsf{Recover}(c, K_s, PK_i)$: *This is a deterministic algorithm run by the sender that takes in the ciphertext, the sender's secret recovery key, the receiver's public key, and returns the message* m. *The algorithm proceeds as follows:*

   (a) *Retrieve the stored* $c'' = c||c'$, *and computes* $\kappa \leftarrow \mathsf{DEMDecrypt}(K_s, c')$.

   (b) *Compute* $m \leftarrow \mathsf{DEMDecrypt}(\kappa, c)$.

□

We could, in principle, plaintext-randomize the symmetric encryption part as well, by sampling message m's plaintext-space, and replacing m with a random string of the same length, but that

would then be equivalent to the well-known model of real-or-random (ROR) security. The point of plaintext randomization is to replace a public key scheme (PKE) that is internal to some larger operation - essentially the PKE ciphertexts give us something that is incidental to the answer that is wanted, but are not what we are ultimately interested in.

CHAPTER 5

ANALYSIS AND PROOFS

In this section, we introduce the plaintext randomization lemma of Tate *et al.* [10], and propose extensions to it. The original plaintext randomization lemma was for a single sender and receiver PKE game, since we are working with single sender and multiple receivers PKE games, we require that the plaintext randomization lemma be extended to the multiple receivers model.

Lemma 1. *(Plaintext randomization lemma for a single receiver [10]) Let* G *be a game that makes* sk*-oblivious use of a plaintext-samplable public key encryption scheme* $\mathfrak{S}$*, and let* $\mathfrak{S}_{\text{-rand}}$ *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,*

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

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

Lemma 2. *(Plaintext randomization lemma for multiple receivers)*

Let G *be a game that makes* sk*-oblivious use of* n *plaintext-samplable public key encryption schemes,* $\mathfrak{S}_1, \cdots, \mathfrak{S}_n$*, and let* $\mathfrak{S}_{\text{-rand}1}, \cdots, \mathfrak{S}_{\text{-rand}n}$ *be the plaintext randomization of* $\mathfrak{S}$*. Then, for any probabilistic adversary* A *that plays the* $\mathsf{G}^{\mathfrak{S}_1}, \cdots, \mathsf{G}^{\mathfrak{S}_n}$*, so that the total game-playing time of* A *is bounded by* t*, the number of calls to* $\mathfrak{S}_1.\mathsf{KeyGen}, \cdots, \mathfrak{S}_n.\mathsf{KeyGen}$ *is bounded by* m*, and the total number of encryption and decryption requests for any individual key is bounded by* q_e *and* q_d *respectively, the advantage of* A *is defined by:*

$$|Adv_{A,\mathsf{G}^{\mathfrak{S}_1,\cdots,\mathfrak{S}_n}} - Adv_{A,\mathsf{G}^{\mathfrak{S}\text{-rand}_1,\cdots,\mathfrak{S}\text{-rand}_n}}| \leq 2\, Adv_{\text{PK-MU}_m^{\mathfrak{S}_1,\cdots,\mathfrak{S}_n}}(t', q_e, q_d),$$

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

Proof. Let A be an adversary for game G, we need to construct an adversary A' for PK-MU$^{\mathfrak{S}_1}_{\mathfrak{S}_n}$, so that A' converts A into an adversary that attacks the multi-user CCA2 security of the underlying PKE scheme $\mathfrak{S}_1, \cdots, \mathfrak{S}_n$. First A' calls PK-MU$^{\mathfrak{S}_1,\mathfrak{S}_n}$.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' then simulates the original adversary A and the game oracle G, replacing G's use of PKEs $\mathfrak{S}_1, \cdots, \mathfrak{S}_n$ with a specially constructed stateful PKE scheme $\tilde{\mathfrak{S}_1}, \cdots, \tilde{\mathfrak{S}_n}$, each keyed with $(pk_1, \cdots, pk_n)$, counter m, as well as PK-MU$^{\mathfrak{S}_1,\cdots,\mathfrak{S}_n}$. For any $i \in \tilde{\mathfrak{S}_1}, \cdots, \tilde{\mathfrak{S}}n$, $\mathfrak{S}_1, \cdots, \mathfrak{S}_n$ provide the three PKE functions:

1. $\tilde{\mathfrak{S}_i}$.KeyGen(1^λ): If $m = n$, i.e., we have already used n keypairs, the operation returns $\perp$ and fails. Else, $\tilde{\mathfrak{S}_i}$ increments i and returns (pk_i, pk_i). The fact that the "secret key" is really the public key does not matter as G's use of $\mathfrak{S}_1, \cdots, \mathfrak{S}_n$ is sk-oblivious.

2. $\tilde{\mathfrak{S}_i}$.Encrypt(pk_i, p): For a valid public key pk_i, $\mathfrak{S}_i$ computes random plaintext $r = \mathfrak{S}_i$.PTSample $(pk_i, p), c =$ PK-MU$^{\mathfrak{S}_1\cdots,\mathfrak{S}_n}$.PEncrypt$(i, p, r)$. The tuple (pk_i, c, p) is saved in $\mathfrak{S}_i$'s state and c is returned.

3. $\tilde{\mathfrak{S}_i}$.Decrypt(pk_i, pk_i, c): The decrypt function checks to see if $\mathfrak{S}_i$'s internal state contains a tuple (pk_i, c, p) for some p and if such a tuple os found p is returned. Else $p =$ PK-MU$^{\mathfrak{S}_1,\cdots,\mathfrak{S}_n}$. Decrypt$(i, c)$ is returned.

All calls to PK-MU$^{\mathfrak{S}_1\cdots,\mathfrak{S}_n}$.PEncrypt store a tuple that includes the returned ciphertext and the $\tilde{\mathfrak{S}_i}$.Decrypt function never calls the PK-MU$^{\mathfrak{S}_1,\cdots,\mathfrak{S}_n}$.Decrypt oracle with such a ciphertext, so all decrypt calls will succeed.

A' will simulate $G^{\tilde{\mathfrak{S}_1},\cdots,\tilde{\mathfrak{S}_n}}$ (note that A') will have to simulate all n of them). Finally, A' will output its final result, a for game G. At this point, A' will call G.IsWinner(a) to check if A's output wins game $G^{\tilde{\mathfrak{S}_i}}$. Based on this, A' outputs its guess b' for PK-MU$^{\mathfrak{S}_1,\cdots,\mathfrak{S}_n}$'s secret bit b. That is, A' needs to output a guess for each of the $\tilde{\mathfrak{S}_1}, \cdots, \tilde{\mathfrak{S}_n}$:

- If A wins $G^{\tilde{\mathfrak{S}_i}}$, A' outputs $b' = 0$

- If A loses $g^{\tilde{\mathfrak{S}}_i}$, A' outputs guess $b' = 1$.

So, A' wins if A wins $G^{\tilde{\mathfrak{S}}_i}$ and $b = 0$, or if A loses and $b = 1$.

$$\begin{aligned} P(A' \text{ wins}) = &\frac{1}{2}P(A \text{ wins } G^{\tilde{\mathfrak{S}}_1}|b=0) + \frac{1}{2}P(A \text{ loses } G^{\tilde{\mathfrak{S}}_1}|b=1) \\ &+ \frac{1}{2}P(A \text{ wins } G^{\tilde{\mathfrak{S}}_2}|b=0) + \frac{1}{2}P(A \text{ loses } G^{\tilde{\mathfrak{S}}_2}|b=1) \\ &+ \cdots \\ &+ \frac{1}{2}P(A \text{ wins } G^{\tilde{\mathfrak{S}}_n}|b=0) + \frac{1}{2}P(A \text{ loses } G^{\tilde{\mathfrak{S}}_n}|b=1) \end{aligned} \tag{5.1}$$

While A' does not know the bit b, the construction of each $\tilde{\mathfrak{S}}_i$ is such that when $b = 0$ the game played by A is exactly $G^{\tilde{\mathfrak{S}}_i}$, and when $b = 1$, the game played by A is exactly $G^{\mathfrak{S}\text{-rand}_i}$. Simplifying Equation 5.1, we get:

$$\begin{aligned} P(A' \text{ wins}) &= \frac{1}{2}P(A \text{ wins } G^{\mathfrak{S}_i}) + \frac{1}{2}P(A \text{ loses } G^{\mathfrak{S}\text{-rand}_i}) \\ &= \frac{1}{2}P(A \text{ wins } G^{\mathfrak{S}_i}) + \frac{1}{2}(1 - P(A \text{ wins } G^{\mathfrak{S}\text{-rand}_i})) \\ &= \frac{1}{2} + \frac{1}{2}P(A \text{ wins } G^{\mathfrak{S}_i}) - \frac{1}{2}P(A \text{ wins } G^{\mathfrak{S}\text{-rand}_i}) \end{aligned} \tag{5.2}$$

$$\begin{aligned} P(A' \text{ wins}) = &\frac{1}{2} + \frac{1}{2}P(A \text{ wins } G^{\mathfrak{S}_1}) - \frac{1}{2}P(A \text{ wins } G^{\mathfrak{S}\text{-rand}_1}) \\ &+ \frac{1}{2} + \frac{1}{2}P(A \text{ wins } G^{\mathfrak{S}_2}) - \frac{1}{2}P(A \text{ wins } G^{\mathfrak{S}\text{-rand}_2}) \\ &+ \cdots \\ &\frac{1}{2} + \frac{1}{2}P(A\, wins G^{\mathfrak{S}_n}) - \frac{1}{2}P(A \text{ wins } G^{\mathfrak{S}\text{-rand}_n}) \end{aligned} \tag{5.3}$$

By summing up the n leading 1/2's and adding up the n probabilities, we get

$$P(A' \text{ wins}) = \frac{1}{2^n} + \frac{1}{2}[P(A \text{ wins } G^{\mathfrak{S}_1}) + P(A \text{ wins } G^{\mathfrak{S}_2}) + \cdots + P(A \text{ wins } G^{\mathfrak{S}_n})] - \frac{1}{2}[P(A \text{ wins } G^{\mathfrak{S}\text{-rand}_1}) + P(A \text{ wins } G^{\mathfrak{S}\text{-rand}_2}) + \cdots + P(A \text{ wins } G^{\mathfrak{S}\text{-rand}_n})] \quad (5.4)$$

By definition, $\text{Adv}_{A',\text{PK-MU}^{\mathfrak{S}_1\cdots,\mathfrak{S}_n}} = |P(A' \text{ wins } - \frac{1}{2})|$, and since A' only does some simple lookups in addition to its simulation of A and G, which induces at most q_e, q_d encryption and decryption requests respectively, we can bound $\text{Adv}_{A',\text{PK-MU}^{\mathfrak{S}_1,\cdots,\mathfrak{S}_n}}$ by $\text{Adv}_{\text{PK-MU}^{\mathfrak{S}_1,\cdots,\mathfrak{S}_n}}(t', q_e, q_d)$, where $t' = t + O(\log(n \cdot q_e))$. It follows that:

$$|\frac{1}{2}P(A \text{ wins } G^{\mathfrak{S}_i}) - \frac{1}{2}P(A \text{ wins } G^{\mathfrak{S}\text{-rand}_i})| \leq \text{Adv}_{\text{PK-MU}^{\mathfrak{S}_i}}(t', q_e, q_d)$$

so,

$$|P(A \text{ wins } G^{\mathfrak{S}_i}) - P(A \text{ wins } G^{\mathfrak{S}\text{-rand}_i})| \leq 2\text{Adv}_{\text{PK-MU}^{\mathfrak{S}_i}}(t', q_e, q_d) \quad (5.5)$$

The left side of Equation 5.5 can be re-written as:

$$\begin{aligned} |P(A \text{ wins } G^{\mathfrak{S}_i}) - P(A \text{ wins } G^{\mathfrak{S}\text{-rand}_i}) = \frac{1}{2^n} &+ \frac{1}{2}[P(A \text{ wins } G^{\mathfrak{S}_1}) - P(A \text{ wins } G^{\mathfrak{S}\text{-rand}_1})] \\ &+ \frac{1}{2}[P(A \text{ wins } G^{\mathfrak{S}_2}) - P(A \text{ wins } G^{\mathfrak{S}\text{-rand}_2})] \\ &+ \cdots \\ &+ \frac{1}{2}[P(A \text{ wins } G^{\mathfrak{S}_n}) - P(A \text{ wins } G^{\mathfrak{S}\text{-rand}_n})] \end{aligned} \quad (5.6)$$

so,

$$P(A' \text{ wins}) = \frac{1}{2^n} + \frac{1}{2}[P(A \text{ wins}, G^{\mathfrak{S}_1}) - P(A \text{ wins } G^{\mathfrak{S}\text{-rand}_1}) + \cdots + P(A \text{ wins } G^{\mathfrak{S}_n}) - P(A \text{ wins } G^{\mathfrak{S}\text{-rand}_n})] \tag{5.7}$$

Comparing the advantage of A w.r.t. $G^{\mathfrak{S}}$ and $G^{\mathfrak{S}\text{-rand}}$, we get:

$$|\text{Adv}_{A,G^{\mathfrak{S}_i}} - \text{Adv}_{A,G^{\mathfrak{S}\text{-rand}_i}}| = |P(A \text{ wins } G^{\mathfrak{S}_i}) - \frac{1}{2}| - |P(A \text{ wins } G^{\mathfrak{S}\text{-rand}_i}) - \frac{1}{2}|$$

so,

$$|\text{Adv}_{A,G^{\mathfrak{S}_i}} - \text{Adv}_{A,G^{\mathfrak{S}\text{-rand}_i}}| \leq |P(A \text{ wins } G^{\mathfrak{S}_i}) - P(A \text{ wins } G^{\mathfrak{S}\text{-rand}_i})| \tag{5.8}$$

Combining Equation 5.5 and Equation 5.8, we get:

$$|\text{Adv}_{A,G^{\mathfrak{S}_i}} - \text{Adv}_{A,G^{\mathfrak{S}\text{-rand}_i}}| \leq 2\text{Adv}_{\text{PK-MU}^{\mathfrak{S}_i}}(t', q_e, q_d)$$

□

□

We now present a proof that our sender recovery scheme (PKE-SR) with plaintext randomization is CCA2-secure, if the PKE scheme and SKE scheme used are both CCA2-secure. Intuitively, we just need to reduce any attack on the PKE-SR scheme to an attack on the SKE scheme.

Theorem 1. *Let A be an adversary that attacks the CCA2 security of our PKE-SR scheme, which uses public-key encryption scheme P as the KEM, and symmetric-key encryption scheme S as the DEM. If A runs in time t in a game that uses at most n keypairs, and performs at most q_e encryptions and q_d decryptions, then*

$$Adv_{A,\textit{PK-MU}_n^{\textit{PKE-SR}}} \leq 2\, Adv_{\textit{PK-MU}_n^P}(t', q_e, q_d) + Adv_{\textit{SK-MU}_{q_e n}^S}(t', 1, q_d)$$

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

Proof. The main idea behind the proof is that in a plaintext-randomized hybrid encryption scheme, we replace the public key encryption scheme with a plaintext-randomized public-key encryption scheme, such that the encrypted session key (encrypted by the KEM) has no relation to the real session key. Hence the KEM and DEM parts of the hybrid encryption scheme are completely unrelated to each other, and the advantage of an adversary in the hybrid encryption scheme can be reduced to, and bounded by the advantage of the adversary in the symmetric encryption scheme that uses the session key to encrypt a message. Beyond this, the sender's recovery scheme just uses the sender's recovery key to recover the session key, which in turn is used to recover the message; this process only uses symmetric keys.

Let us consider the plaintext randomized version of our PKE-SR scheme, PKE-SR-rand. In the plaintext randomized version, we will generate keypairs for the sender and receiver, and the sender will generate a session key, κ. The sender will then generate a ciphertext c_{KEM} which is the encryption of a random string r, generated by sampling the plaintext-space of κ, and $|r| = |\kappa|$. The sender will then encrypt a message m by doing $c_{DEM} = \mathsf{DEMEncrypt}(\kappa, m)$, and send $c = (c_{KEM}, c_{DEM})$ to the receiver. Since c_{KEM} is an encryption of a random string, it has no relation to c_{DEM}.

Let us consider an adversary A that plays PKE-SR-rand. A can easily be turned into an adversary A' that plays the SKE game against S, since A' can just simulate the A by adding encryption of random values for c_{KEM}, and the DEM part will be a symmetric-key encryption of the real message with κ. The sender recovery part will just be a symmetric encryption of the real session key with the sender's recovery key. Hence the probability of A winning the PKE-SR game is the same as the probability of A' winning the SKE game, and we can bound the advantage of A as:

$$Adv_{A,\text{PK-MU}_n^{\text{PKE-SR-rand}}} \leq Adv_{A',\text{SK-MU}_{q_e n}^{S}}(t, 1, q_d)$$

From the plaintext randomization lemma we get:

$$|Adv_{A,\text{PK-MU}_n^{\text{PKE-SR}}} - Adv_{A,\text{PK-MU}_n^{\text{PKE-SR-rand}}}| \leq 2\, Adv_{\text{PK-MU}_n^P}(t', q_e, q_d)$$

and hence,

$$Adv_{A,\text{PK-MU}_n^{\text{PKE-SR}}} \leq 2\, Adv_{\text{PK-MU}_n^P}(t', q_e, q_d) + Adv_{\text{SK-MU}_{q_e n}^S}(t', 1, q_d)$$

□

□

CHAPTER 6

PRACTICAL INSTANTIATIONS

In this section, we describe an instantiation of our PKE-SR scheme using the classic Cramer-Shoup KEM/DEM scheme [7]. Cramer and Shoup's work [7], was the first work to rigorously establish the security of hybrid encryption. The idea of hybrid encryption has been around since the 1980s, primarily due to the inefficiency of public key encryption, and intuition tells us that if the symmetric key scheme and the public key scheme used in hybrid encryption are both secure in some sense (e.g., CCA2 secure), then a hybrid encryption scheme that composes them should be secure as well. In spite of this, the security of hybrid encryption was tricky to formally establish until the work of Cramer and Shoup, who introduced the primitives of key encapsulation mechanism (KEM) for generating and encrypting the session key, and data encapsulation mechanism (DEM) for encrypting data with the session key. Their work was instrumental in establishing a rigorous foundation for the analysis of hybrid encryption schemes. We first review some preliminary definitions from [7]. The original Cramer-Shoup scheme is given in Appendix B for reference.

Computational group scheme: A computational group scheme $\mathcal{G}$ specifies a sequence S_λ, where $\lambda \in \mathbb{Z}^+$. For every value of λ, S_λ is a probability distribution of group description, where a group description Γ specifies a finite abelian group $\hat{G}$, along wth a prime-order subgroup G, a generator g of G, and the order q of G. Let $\Gamma[\hat{G}, G, g, q]$ indicate that Γ specifies $\hat{G}, G, g$ and q. The group scheme also provides several algorithms to test membership and group properties, such as closure, existence of identity element, inverse element, associativity.

Target collision hash function: Let $k \in \mathbb{Z}^+$, such that k is constant, and let $\mathcal{G}$ be a computational group scheme, specifying a sequence S_λ of group distributions, where $\lambda \in \mathbb{Z}^+$ is a security parameter. Then HF is a k-ary group hashing scheme that specifies two algorithms:

- A family of key spaces indexed by $\lambda \in \mathbb{Z}^+$ and $\Gamma \in |S_\lambda|$. Each such key space is a probability space on bit strings denoted by $\mathsf{HF.KeySPace}_{\lambda,\Gamma}$. There must exist a probabilistic, polynomial-time algorithm whose output distribution on input 1^λ and Γ is equal to $\mathsf{HF.KeySpace}_{\lambda,\Gamma}$.

- A family of hash functions indexed by $\lambda \in \mathbb{Z}^+$, $\Gamma[\hat{G}, G, g, q] \in [S_\lambda]$, and $hk \in [\mathsf{HF.KeySpace}_{\lambda,\Gamma}]$, and $\rho \in G^k$, outputs $\mathsf{HF}_{\mathsf{hk}}^{\lambda,\Gamma}(\rho)$.

The target collision resistance assumption for HF is then this: for every probabilistic polynomial-time algorithm A, the function $Adv\mathrm{TCR}_{\mathsf{HF,A}}(\lambda)$ is negligible in λ.

$$Adv\mathrm{TCR}_{\mathsf{HF},A}(\lambda|\Gamma) = Pr[\rho in G^k \wedge \rho \neq \rho^* \wedge \mathsf{HF}_{\mathsf{hk}}^{\lambda,\Gamma}(\rho^*) = \mathsf{HF}_{\mathsf{hk}}^{\lambda,\Gamma}(\rho) : \rho^* \leftarrow G; \mathsf{hk} \leftarrow \mathsf{HF.KeySpace}_{\lambda,\Gamma}; \rho \leftarrow A(1^\lambda, \Gamma, \rho^*, \mathsf{hk})]$$

Key Derivation Functions: Let $\mathcal{G}$ be a computational group scheme, specifying a sequence $(\mathbf{S}_\lambda)$ of group distributions. A key derivation function (KDF), associated with $\mathcal{G}$, specifies two items:

- A family of *key spaces* indexed by $\lambda \in \mathbb{Z}^+$, and $\Gamma \in [\mathbf{S}_\lambda]$. Each such key space is a probability space, denoted $\mathsf{KDF.KeySpace}_{\lambda,\Gamma}$, on bit strings, called derivation keys. There must exist a probabilistic, polynomial time algorithm, whose output distribution on input 1^λ and Γ is equal to $\mathsf{KDF.KeySpace}_{\lambda,\Gamma}$.

- A family of *key derivation functions* indexed by $\lambda \in \mathbb{Z}^+$, $\Gamma[\hat{G}, G, g, q] \in [\mathbf{S}_\lambda]$, and $\mathsf{dk} \in [\mathsf{KDF.KeySpace}_{\lambda,\Gamma}]$, where each such function $\mathsf{KDF}_{\mathsf{dk}}^{\lambda,\Gamma}$ maps a pair $(a, b) \in G^2$ of group elements to a key K. A key k is a bit string of length $\mathsf{KDF.OutLen}(\lambda)$. The parameter $\mathsf{KDF.OutLen}(\lambda)$ should be computable in deterministic polynomial time, given 1^λ. There must exist a deterministic polynomial-time algorithm that on input 1^λ, $\Gamma[\hat{G}, G, g, q] \in [\mathbf{S}_\lambda]$, $\mathsf{dk} \in [\mathsf{KDF.KeySpace}_{\lambda,\Gamma}]$, and $(a, b) \in G^2$, outputs $\mathsf{KDF}_{\mathsf{dk}}^{\lambda,\Gamma}(a, b)$.

The key derivation function security assumption is then this: for all probabilistic, polynomial-time algorithms A, and for all $\lambda \in \mathbb{Z}^+$, the function $Adv\text{Dist}_{\mathsf{KDF},A}$ is negligible in λ:

$$
\begin{aligned}
Adv\text{Dist}_{\mathsf{KDF},A}(\lambda) = |Pr[\tau = 1 : \Gamma \leftarrow \mathbf{S}_\lambda; \mathsf{dk} \leftarrow \mathsf{KDF.KeySpace}_{\lambda,\Gamma}; a, b \leftarrow G; \\
\tau \leftarrow A(1^\lambda, \Gamma, \mathsf{dk}, a, \mathsf{KDF}^{\lambda,\Gamma}_{\mathsf{dk}}(a, b))] - \\
Pr[\tau = 1 : \Gamma \leftarrow \mathbf{S}_\lambda; \mathsf{dk} \leftarrow \mathsf{KDF.KeySpace}_{\lambda,\Gamma}; a \leftarrow G; K \leftarrow \{0,1\}^{\mathsf{KDF.OutLen}(\lambda)}; \\
\tau \leftarrow A(1^\lambda, \Gamma, \mathsf{dk}, a, K)]|
\end{aligned}
$$

Definition 8. ***(PKE-SR using plaintext randomized Cramer-Shoup (CS) scheme)***

- $(PK, SK, K_s) \leftarrow \mathsf{CS.KeyGen}(1^\lambda)$*: This is a randomized algorithm run by the sender and receiver individually to generate their respective keys. The key generation process proceeds as follows:*
  - $\Gamma[\hat{G}, G, g, q] \leftarrow \hat{S}(1^\lambda)$
  - $\mathsf{hk} \leftarrow \mathsf{HF.KeySpace}_{\lambda,\Gamma}$
  - $\mathsf{dk} \leftarrow \mathsf{KDF.KeySpace}_{\lambda,\Gamma}$
  - $\mathsf{K_s} \leftarrow \mathsf{KDF.KeySpace}_{\lambda,\Gamma}$
  - $z_1, z_2 \leftarrow \mathbb{Z}_q; h \leftarrow g^{z_1}\hat{g}^{z_2}$
  - *Set* $PK = (\Gamma, \mathsf{hk}, \mathsf{dk}, h)$, $SK = (\Gamma, \mathsf{hk}, \mathsf{dk}, z_1, z_2)$. *Set* K_s *as the sender's recovery key.*
- $(c = (c_{KEM}, c_{DEM})) \leftarrow \mathsf{CS.Encrypt}(PK, K_s, m)$*: This is a randomized algorithm run by the sender that takes as input the receiver's public key, sender's recovery key, a message* m*, and outputs a ciphertext. The algorithm proceeds as follows:*
  - *Compute* $u \leftarrow \mathbb{Z}_q$*,* $a \leftarrow g^u$*,* $b \leftarrow h^u$*,* $\kappa \leftarrow \mathsf{KDF}^{\lambda,\Gamma}_{dk}(a, b)$.
  - *Sample the key-space of* κ*, and produce a random string:* $\hat{\kappa} \leftarrow \mathsf{CS.PTSample}(PK, \kappa)$*, such that* $|\hat{\kappa}| = |\kappa|$.

  - *Compute* $c_{KEM} = \mathsf{CS.KEMEncrypt}(PK, K_s, \hat{\kappa})$. *Store* (PK, c_{KEM}, κ) *in internal state, if it does not already exist.*
  - *Compute* $c_{DEM} \leftarrow \mathsf{CS.DEMEncrypt}(\kappa, m)$. *Compute* $c' = \mathsf{CS.DEMEncrypt}(\kappa, K_s)$, *set* $c'' = c||c'$.
  - *Send* c *to the receiver, and store* c''.

- $\{m, \perp\} \leftarrow \mathsf{CS.Decrypt}(SK, c = (c_{KEM}, c_{DEM}))$*: This is a deterministic algorithm run by the receiver that takes in the receiver's secret key and the ciphertext, and outputs the message* m*. The algorithm proceeds as follows:*
  - *Check if the internal state of the KEM scheme contains a tuple* (PK, c_{KEM}, κ). *If such a tuple is found, then* κ *is returned as the session key. Else,* $\kappa \leftarrow \mathsf{CS.DEMDecrypt}(SK, c_{KEM})$ *is returned.*
  - *Compute* $b = a^{z_1}\hat{a}^{z_2}$. *Compute* $\kappa \leftarrow \mathsf{KDF}_{dk}^{\lambda,\Gamma}(a, b)$.
  - *Decrypt the message* m: $m \leftarrow \mathsf{CS.DEMDecrypt}(\kappa, c_{DEM})$.

- $m \leftarrow \mathsf{CS.Recover}(c'')$*: This is a deterministic algorithm run by the sender when the sender wants to recover message* m. *The sender retrieves* $c'' = c||c'$, *and computes* $\kappa = \mathsf{CS.DEMDecrypt}(K_s, c')$. *Sender then does* $m \leftarrow \mathsf{CS.DEMDecrypt}(K_s, c_{DEM})$.

□

Theorem 2. *Let* A *be a probabilistic, polynomial-time adversary that attacks the CCA2 security of the Cramer-Shoup hybrid encryption scheme,* $\mathsf{CS}(P, S)$*, where* P *is the public-key encryption scheme, and* S *is the shared-key encryption scheme used by the* CS *scheme. If* A *runs in time* t *in a game that used at most* n *key-pairs, and performs at most* q_e *encryptions and* q_d *decryptions, then,*

$$Adv_{A,\text{PK-MU}^{\mathsf{CS}}} \leq 2Adv_{\text{PK-MU}^{P_n}}(t', q_e, q_d) + Adv_{\text{SK-MU}^{S}_{q_e n}}(t', 1, q_d)$$

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

Proof. In our construction, the hybrid encryption scheme uses plaintext randomization to cut out any use of the encrypted key from the symmetric encryption scheme. Hence, we can use a general purpose public-key encryption scheme, along with a shared-key encryption scheme. Since the "encryptions" of the ciphertexts in the key encapsulation stage are just encryptions of random values, the security of the hybrid encryption scheme can be reduced to an attack on S. Since the sender recovery part also uses the same symmetric key, the probability of the adversary, A, winning the CS game can be bounded by the probability of the adversary winning the symmetric-key game of S:

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

And from plaintext randomization lemma 2, we know that:

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

and so,

$$Adv_{A,\text{PK-MU}^{\mathsf{CS}}} \leq 2Adv_{\text{PK-MU}^{P_n}}(t', q_e, q_d) + Adv_{\text{SK-MU}^{S}_{q_e n}}(t', 1, q_d)$$

□

□

Put, in simple terms, the security of the CS hybrid encryption scheme that uses plaintext randomization is adaptive CCA2-secure, of its constituent PKE and SKE are adaptive CCA2-secure. The corollary follows.

Corollary 1. *If P is an adaptive CCA2-secure PKE scheme, and S is an adaptive CCA2-secure SKE scheme, then CS-rand, which is the plaintext randomized version of the Cramer-Shoup hybrid encryption scheme, CS, is an adaptive CCA2-secure hybrid encryption scheme.*

CHAPTER 7

CONCLUSION

We have presented a protocol for public key encryption with sender recovery, which helps a sender independently recover a plaintext message encrypted under the public key of a receiver, without the help of the receiver, and without having to store a local copy. Our scheme works for multiple receivers, and the sender would just have to store one recovery key for multiple receivers. This has potential applications in untrusted third party data storage, file sharing and collaboration services, and e-mail recovery services. Our protocol is CCA2 secure, and is based on KEM/DEM-based hybrid encryption and a technique called plaintext randomization. Compared to prior work in this area, our scheme does not require the KEM/DEM to be CCA2 secure, has minimal assumptions, and results in simple, intuitive proofs.

BIBLIOGRAPHY

[1] Bellare, M., Boldyreva, A., Micali, S.: Public-key encryption in a multi-user setting: Security proofs and improvements. In: Advances in Cryptology - EUROCRYPT. pp. 259–274 (2000)

[2] Bellare, M., Desai, A., Jokipii, E., Rogaway, P.: A concrete security treatment of symmetric encryption. In: 38th Annual Symposium on Foundations of Computer Science, FOCS. pp. 394–403 (1997)

[3] Bellare, M., Miner, S.K.: A forward-secure digital signature scheme. In: Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings. pp. 431–448 (1999)

[4] Bellare, M., Yee, B.S.: Forward-security in private-key cryptography. In: Topics in Cryptology - CT-RSA 2003, The Cryptographers' Track at the RSA Conference 2003, San Francisco, CA, USA, April 13-17, 2003, Proceedings. pp. 1–18 (2003)

[5] Blömer, J., Liske, G.: Direct chosen-ciphertext secure attribute-based key encapsulations without random oracles. IACR Cryptology ePrint Archive 2013, 646 (2013)

[6] Choi, K.Y., Cho, J., Hwang, J.Y., Kwon, T.: Constructing efficient PAKE protocols from identity-based KEM/DEM. vol. 2015, p. 606 (2015)

[7] Cramer, R., Shoup, V.: Design and analysis of practical public-key encryption schemes secure against adaptive chosen ciphertext attack. vol. 33, pp. 167–226 (2003)

[8] Galindo, D., Großschädl, J., Liu, Z., Vadnala, P.K., Vivek, S.: Implementation and evaluation of a leakage-resilient elgamal key encapsulation mechanism. vol. 2014, p. 835 (2014)

[9] Liu, S., Paterson, K.G.: Simulation-based selective opening CCA security for PKE from key encapsulation mechanisms. In: Public-Key Cryptography - PKC 2015 - 18th IACR International Conference on Practice and Theory in Public-Key Cryptography, Gaithersburg, MD, USA, March 30 - April 1, 2015, Proceedings. pp. 3–26 (2015)

[10] Tate, S.R., Vishwanathan, R., Weeks, S.: Encrypted secret sharing and analysis by plaintext randomization. In: 16^{th} Information Security Conference ISC. pp. 49–65 (2013)

[11] Wei, P., Zheng, Y.: Efficient public key encryption admitting decryption by sender. In: Public Key Infrastructures, Services and Applications - 9th European Workshop, EuroPKI. pp. 37–52 (2012)

[12] Wei, P., Zheng, Y.: On the construction of public key encryption with sender recovery. Int. J. Found. Comput. Sci. 26(1), 1–32 (2015)

[13] Wei, P., Zheng, Y., Wang, X.: Public key encryption for the forgetful. In: Cryptography and Security. pp. 185–206 (2012)

APPENDICES

APPENDIX A
NEW PKE-SR SCHEME

Definition 9. *(New PKE-SR scheme)*

1. $(PK_r, SK_r, K_s) \leftarrow \mathsf{KeyGen}(1^\lambda)$*: This is a randomized algorithm, run individually by both parties, that outputs a public/secret key-pair,* (PK_r, SK_r) *for the receiver and a secret recovery key,* K_s *for the sender.*

2. $c \leftarrow \mathsf{Encrypt}(K_s, PK_r, m)$*: This is a randomized algorithm run by the sender, that takes as input a message* m *to be encrypted, the receiver's public key* PK_r*, the sender's recovery key outputs a ciphertext c. The sender computes* $(\kappa, c_{KEM}) \leftarrow \mathsf{KEMEncrypt}(K_s, PK_r, \tau = \{0,1\}^\lambda)$*, and encrypts the message* m*:* $c_{DEM} \leftarrow \mathsf{DEMEncrypt}(\kappa, m)$*. The sender then sets* $c = (c_{KEM}, c_{DEM})$[1]*.*

   For enabling recovery at a later stage, the sender computes $c' = \mathsf{DEMEncrypt}(K_s, \kappa)$*, sets* $c'' = c||c'$*, and stores* c''*.*

3. $m \leftarrow \mathsf{Decrypt}(c, SK_r)$*: This is a deterministic algorithm run by the receiver that takes as input a ciphertext c, the receiver's secret key, and outputs the message* m*. The receiver retrieves the session key,* $\kappa \leftarrow \mathsf{KEMDecrypt}(c_{KEM}, SK_r)$*, and obtains the message* m*,* $m \leftarrow \mathsf{DEMDecrypt}(\kappa, c_{DEM})$*.*

4. $m \leftarrow \mathsf{SR.Recover}(c'', K_s, PK_r)$*: This is a deterministic algorithm run by the sender that takes as input a ciphertext, the sender's secret recovery key, the receiver's public key, and returns the message* m*. The sender retrieves the stored* $c||c' = c''$*, computes* $\kappa \leftarrow \mathsf{DEMDecrypt}(K_s, c')$*, and finally computes* $m \leftarrow \mathsf{DEMDecrypt}(\kappa, c)$*.*

[1] For clarity of presentation, we omit the details of the internal randomness generated by the encryption algorithm.

APPENDIX B
CRAMER-SHOUP KEM SCHEME

Definition 10. *Cramer-Shoup KEM Scheme*

- $(PK, SK) \leftarrow \mathsf{KeyGen}(1^\lambda)$*: This is a randomized algorithm that generates a public/secret keypair on input* 1^λ*. On input* 1^λ*, it computes* $\Gamma[\hat{G}, G, g, q] \leftarrow \hat{S}^\lambda$*,* $\mathsf{hk} \leftarrow \mathsf{HF.KeySpace}_{\lambda,\Gamma}$*,* $\mathsf{dk} \leftarrow \mathsf{KDF.KeySpace}_{\lambda,\Gamma}$*,* $w \leftarrow \mathbb{Z}_q^*$*,* $x_1, x_2, y_1, y_2, z_1, z_2 \leftarrow \mathbb{Z}_q$*,* $\hat{g} \leftarrow g^w$*,* $e \leftarrow g^{x_1}\hat{g}^{x_2}$*,* $f \leftarrow g^{y_1}\hat{g}^{y_2}, h \leftarrow g^{z_1}\hat{g}^{z_2}$*. Sets the public key as* $PK = (\Gamma, \mathsf{hk}, \mathsf{dk}, \hat{g}, e, f, h)$*, and secret key as* $SK = (\Gamma, \mathsf{hk}, \mathsf{dk}, x_1, x_2, y_1, y_2, z_1, z_2)$*.*

- $(K, \psi) \leftarrow \mathsf{Encrypt}(PK, 1^\lambda)$*: This is a randomized algorithm that generates a symmetric key,* K*, and the encryption of* K*,* ψ*. It computes* $u \leftarrow \mathbb{Z}_q$*,* $a \leftarrow g^u$*,* $\hat{a} \leftarrow \hat{g}^u$*,* $b \leftarrow h^u$*,* $K \leftarrow \mathsf{KDF}_{\mathsf{dk}}^{\lambda,\Gamma}(a, b)$*,* $v \leftarrow \mathsf{HF}_{\mathsf{hk}}^{\lambda,\Gamma}(a, \hat{a})$*,* $d \leftarrow e^u f^{uv}$*. Output* K*,*$\psi = (a, \hat{a}, d)$*.*

- $\{K, \perp\} \leftarrow \mathsf{Decrypt}(SK, \psi)$*. This is a deterministic algorithm that takes in as input a ciphertext and a secret key, and produces the symmetric key,* K*. If* $\psi \neq (a, \hat{a}, d) \in \hat{G}^3$*, return* $\perp$*. Else if* $\{a, \hat{a}\} \notin G$*, return* $\perp$*. Else compute* $v = \mathsf{HF}_{\mathsf{hk}}^{\lambda,\Gamma}(a, \hat{a})$*. If* $d \neq a^{x_1+y_1 v}\hat{a}^{x_2+y_2 v}$*, return* $\perp$*. Else compute* $b \leftarrow a^{z_1}\hat{a}^{z_2}$*. Compute* $K \leftarrow \mathsf{KDF}_{\mathsf{dk}}^{\lambda,\Gamma}(a, b)$*, and output* K*.*