Designing Generic Asymmetric Key Cryptosystem with Message Paddings

by

Tarun Kumar Bansal

Indraprastha Institute of Information Technology, Delhi
and
Queensland University of Technology, Australia

Supervisors: Dr. Donghoon Chang (IIITD)
Dr. Josef Pieprzyk (QUT)
Dr. Somitra Sanadhya (IIT-Ropar, formerly at IIITD)
Dr. Xavier Boyen (QUT)

October 2017

Designing Generic Asymmetric Key Cryptosystem with Message Paddings

by

Tarun Kumar Bansal

Thesis submitted in accordance with the joint regulations for the Degree of Doctor of Philosophy by and between

Indraprastha Institute of Information Technology, Delhi
and
Queensland University of Technology, Australia

October 2017

Keywords

Public-key cryptography, Arbitrary long message, Hybrid Encryption, Sponge, permutation, Weakly Secure, Digital Signatures, CCA-secure, OAEP, padding.

Abstract

RSA-OAEP is being used in PKCS #1 2.0 standard for a long time. OAEP (optimal asymmetric encryption padding) provides security strength to RSA and other deterministic one-way asymmetric primitives (trapdoor one-way permutations). OAEP has been found to be useful in case of hybrid encryption, signcryption, hybrid signcryption and also as randomness recovery scheme. With time, several proposals modifying OAEP were published in the literature. These proposals give different OAEP versions which differ regarding efficiency, provable security, compatibility with a type of asymmetric one-way cryptosystem (deterministic or probabilistic), extending the use of OAEP in other applications, etc.

Our work helps in understanding the development of OAEP framework and its use. As part of our contribution, we describe a different kind of message padding which works as an alternative of OAEP type scheme. This new message padding scheme is based on iterated Sponge permutation structure. Usage of famous Sponge permutation structure comes from symmetric cryptography where iterated permutation as Sponge functions has provided a great feature to align security and efficiency. We call our scheme *Sponge based asymmetric encryption padding* (SpAEP). Our scheme achieves semantic security under chosen ciphertext attack (IND-CCA) using any trapdoor one-way permutation in the ideal permutation model for arbitrary length messages. This IND-CCA security is considered as highest and strongest security notion, whereas one-wayness security notion is weaker one. We also propose a key encapsulation mechanism for hybrid encryption using SpAEP with any trapdoor one-way permutation. SpAEP utilizes the permutation model efficiently in the setting of public key encryption in a novel manner.

A primary limitation with the OAEP-type schemes is their incompatibility with a probabilistic asymmetric one-way secure cryptosystem (e.g., ElGamal). We study the reasons behind this limitation and are able to extend the scope of

usage from deterministic (e.g., RSA) to probabilistic (e.g., ElGamal) functions along with efficiency improvements in SpAEP. We denote new modified Sponge based padding as SpPad–Pe where SpPad–Pe stands for Sponge based Padding (SpPad) with asymmetric one-way cryptosystem (Pe).

The concept and techniques which are used as a base for constructing Sponge based message padding, also result in a strongly secure generic asymmetric encryption scheme using weakly secure asymmetric cryptosystem. Instead of using specific Sponge based construction, we introduce a more generic framework to build a CCA-secure PKE, called REAL. REAL stands for Real time CCA-secure Encryption for Arbitrary Long Messages. An asymmetric one-way secure cryptosystem, a one-time secure symmetric encryption scheme and two hash functions are sufficient for this design. Proposed design provides streaming option without compromising other valuable features, compared to previous works.

We exploit versatile nature of Sponge construction into another area of cryptography known as signcryption. The aim of signcryption is to provide both confidentiality and authentication of messages more efficiently than performing encryption and signing independently. "Commit-then-Sign&Encrypt" (CtS&E) composition method allows to perform encryption and signing in parallel. Parallel execution of cryptographic algorithms decreases the computation time needed to signcrypt a message. We put forward the application of sponge structure based message padding as an alternative of commitment scheme in constructing signcryption scheme. We propose a provably secure signcryption scheme using weak asymmetric primitives such as trapdoor one-way encryption and universal unforgeable signature. Using simple tricks, we also demonstrate how different combinations of probabilistic/deterministic encryption and signature schemes following weaker security requirements can be utilized without compromising the security of the scheme. To the best of our knowledge, this is the first signcryption scheme based on sponge structure and offers maximum security using weak underlying asymmetric primitives along with the ability to handle long messages.

This thesis follows a step-by-step formation of efficient and secure cryptosystem, starting from basic to complex structure. This thesis emphasizes the importance of message pre-processing technique and its usage by providing generic and efficient cryptosystem.

Contents

List of Figures

List of Tables

Declaration

The work contained in this joint PhD thesis undertaken between Indraprastha Institute of Information Technology and Queensland University of Technology has not been previously submitted to meet requirements for an award at these or any other higher education institutions. To the best of my knowledge and belief, the thesis contains no material previously published or written by another person except where due reference is made.

Signed:. . . . . **Date:**. . . . .

List of Publications

The following papers have been published or presented, and contain material based on the content of this thesis.

1. Tarun Kumar Bansal, Donghoon Chang, and Somitra Kumar Sanadhya. *Sponge based CCA2 secure asymmetric encryption for arbitrary length message.* In Ernest Foo and Douglas Stebila, editors, Information Security and Privacy - 20th Australasian Conference, ACISP 2015, Brisbane, QLD, Australia, June 29 - July 1, 2015, Proceedings, volume 9144 of Lecture Notes in Computer Science, pages 93-106. Springer, 2015.

2. Tarun Kumar Bansal, Donghoon Chang, and Somitra Kumar Sanadhya. *Sponge based CCA2 secure asymmetric encryption for arbitrary length message (extended abstract). Full paper.* International Journal of Applied Cryptography (IJACT). Editor in Chief: Dr. Yi Mu, Dr. David Pointcheval. Vol. 3 No. 3, pp 262-287, 2017.

Acknowledgements

I am grateful to the many people who have helped me throughout my Ph.D. journey. First, I thank my supervisors, Dr. Donghoon Chang, Dr. Josef Pieprzyk, Dr. Xavier Boyen and Dr. Somitra Sanadhya for their guidance and support over the years. I am not just thankful to my advisors for their patience and moral support, but also for the hard questions which incented me to widen my research from various perspectives.

Besides my advisors, I would like to thank the rest of my internal thesis committee at QUT: Prof. Ed Dawson and Dr. Harry Bartlett, for their insightful comments and encouragement.

I have had many helpful discussions with my colleagues in the Cryptology Research Group at Indraprastha Institute of Information Technology-Delhi (IIITD) and Information Security Discipline of Queensland University of Technology (QUT). I appreciate the joint funding support from IIITD and QUT during this joint Ph.D. program.

I would like to acknowledge my mother, father, wife, both elder brothers and sister-in-law who have supported and encouraged me. Last but not least, I acknowledge all my friends in India and Australia who helped me during my Ph.D. candidature and various travels.

;

Chapter 1

Introduction

Contents

Cryptology is the science and art of secret communications. Historically, cryptology has been used by diplomatic missions and armed forces. However, with the ease of availability and low cost of computing facilities and internet, the domain of cryptology has shifted to non-government uses and to fulfill common needs of individuals. Today, cryptology is used in securing access to internet banking, secure login to websites, secure payment on shops, protecting the integrity of data online, secure computations on cloud, etc.

Cryptology comprises of two broad types of studies - *cryptography* and *cryptanalysis*. While cryptography deals with the design of mechanisms providing certain security goals, cryptanalysis focuses on analyzing these designs with the aim of finding some flaw/weakness in them and violate security goals. These security goals include primary function of cryptographic security namely

1. Confidentiality or Privacy: only intended recipient can see the message.
2. Integrity: message has not been altered in between sender and recipient.
3. Authentication: a process of proving legitimate identity of sender/receiver.
4. Non-repudiation: sender can not deny a message which is sent by him.
5. Key exchange: a method for exchanging keys between sender and receiver.

1.1 Type of Cryptographic Algorithms

1.1.1 Symmetric Cryptography

First type of cryptography started in secret communication is symmetric cryptography. In symmetric cryptographic algorithms, sender and receiver requires the knowledge of one common key which is kept secret from other unauthorized parties. These algorithms accept data (message or plaintext) and key as their inputs and transform them to some other output (ciphertext). The transformed data is then exchanged between the communicating parties. Only the authorized members can recover the plaintext from this ciphertext, for the rest, this ciphertext appears illegible. The primary goal of these algorithms is to provide confidentiality. If the secret key is used only one time for encryption, then such encryption is also known as one-time encryption.

Drawback of symmetric cryptography: The one major problem that held back a general uptake of cryptography for use in business circles was that of exchanging keys. While for many years, governments had established methods of managing keys, business people were not interested in employing circumspect, and perhaps even dangerous, methods of exchanging keys. In the 1960s, this became known as the “key management” problem and it was to be another decade before a viable solution was found.

This key management problem led to open a new type of cryptography, known as asymmetric cryptography or public key cryptography. Basically, in asymmetric cryptography, all communicating parties require to have two keys, one is a publicly known key (for encryption) and other one is a secretly owned private key (for decryption). In this thesis, we will concentrate on asymmetric cryptographic schemes and their provable secure design mechanism. In next section, we briefly describe the history and origin of asymmetric cryptography.

1.1.2 Asymmetric cryptography

In 1976, Whitfield Diffie and Martin Hellman published a paper [51] describing a method of establishing a common key in a secure manner over an insecure channel. The method is based on exponentiation and the fact that exponents can be multiplied in any order with the same result. However, this scheme was useful only for establishing keys and did not actually encrypt data. The search was still on for an encryption scheme that allowed anyone to send an enciphered message to any other person, without pre-establishing keys, such that only the targeted recipient could decrypt the message.

In 1978, the first publicly available method for implementing such a scheme was published by Rivest et al. [94] and is now widely known by the first letter of each of the authors' names as RSA. RSA security is based on the difficulty of factoring large numbers. RSA provides a family of trapdoor one-way permutations, where the secret parameter works as a trapdoor to invert the output, where input and output are of equal bit length.

The ElGamal cryptographic algorithm [57] was invented a few years after the RSA scheme, developing from the PhD thesis of Taher ElGamal, which was awarded in 1984. The underlying idea on which the security is based is quite different from that of RSA. In ElGamal, the target is to determine the exponent in an equation of the form $a = b^x$, where a and b are known. The inventor did not apply for a patent on his scheme. ElGamal provides a family of trapdoor one-way functions, where the secret parameter is a trapdoor used to invert the output, where output is of larger bit size compared to input.

All known public key schemes are far more computationally intensive than symmetric key schemes. For example, a disadvantage of the ElGamal system is that the encrypted message becomes much larger than plaintext, about twice the size of the original text. Similarly, RSA is slower than DES (a symmetric key based cipher) by a factor of about 1000. For this reason, public key schemes are traditionally used only for small messages such as secret keys, whereas symmetric key schemes are retained for sending large messages. Independent from type of encryption, the underlying mathematical formulation needs to be based on a finite system in order to ensure that infinite loops are avoided in computations. A second common feature of symmetric and asymmetric encryption is the use of both an encryption and a decryption key where data is transmitted over an insecure channel. While many public key cryptosystem have been proposed, only a few have withstood the test of time to remain in use today.

Irrespective of many different schemes proposed in literature, all of them use some assumptions, security goals, etc., and each scheme is different from others in terms of type and number of assumptions and security goals. Asymmetric cryptography is primarily preferred for authentication, non-repudiation, and key exchange.

1.1.3 Hash functions

A cryptographic hash function is an algorithm which processes an arbitrary length message into a fixed-length digest or hash code. Hash functions provide one-way cryptography since the message (or plaintext) is not recoverable from digest (or ciphertext). Diffie and Hellman [51] identified the need for a one-way hash function as a building block of a digital signature scheme. The first definitions, analysis, and constructions for cryptographic hash functions can be found in the work of Rabin [91], Yuval [102], and Merkle [76] of the late 1970s. Some design principles for hash function are introduced and discussed in [39, 46, 76].

Hash functions play an important role in both symmetric and asymmetric cryptography for providing integrity and authentication. When a hash function is dependent on a key for its calculation then the output (or digest) is known as message authentication codes (MAC) or tag. By the combination of MAC and symmetric encryption, authenticated encryption (AE) algorithms are constructed which provide both privacy and authentication simultaneously. Hash functions along with asymmetric encryption provide digital signatures for ensuring non-repudiation, integrity, and authentication.

For security proof, hash functions require a theoretical ideal behavior and are considered as ideal randomized black box function also called random oracle [12,33]. In brief, a random oracle is an ideal random function that is publicly available for computation without knowing its internal structure. For each new arbitrary long or fixed length input, the random oracle outputs a fixed length random output from a fixed range. Security proof considering hash functions as random oracle is described as security proof in random oracle model.

1.2 Motivation

1.2.1 Role of message padding in development of asymmetric encryption

Asymmetric encryption has developed over four decades and still growing. Some notable changes in asymmetric encryption are about getting higher security from weakly secure systems, increasing efficiency of asymmetric encryption by combining with symmetric cryptosystem and broader area of applications from encryption to authenticated encryption.

After the introduction of RSA as a first method to build a public key encryption, though RSA partially satisfies security notion of one-wayness it is not suitable as a complete cryptosystem. This unsuitability due to the fact that it is not semantically secure (a.k.a indistinguishability) under chosen plaintext. Given a ciphertext c obtained from some m, an adversary can easily create a another ciphertext c' using public information for which the decryption m' will be tightly related to m. This malleable nature of RSA allows the adversary to make predictable changes in the ciphertext and the underlying plaintext. RSA is deterministic trapdoor one-way permutation, and thus not semantically secure. This semantic insecurity allows an adversary to distinguish between the encryptions of two different messages, simply by encrypting both values himself and comparing the ciphertexts. A message preprocessing technique was introduced for using RSA in practice which also provides semantic security. As part of first standard PKCS1 v1.5, random strings are added to a message, and then this updated message is given as an input to RSA. The resulting system is believed to be secure and provides semantic security. This technique remains widely deployed in many web servers and browsers. While researchers were busy in standardizing the usage of RSA in practice, many different security notions were evolving, like semantic security under chosen ciphertext attack (IND-CCA), which is found to be more suitable for a practical cryptosystem. In 1994, it was realized that current RSA standard PKCS1v1.5 does not satisfy this (IND-CCA) security. Bellare and Rogaway proposed another message pre-processing technique called "Optimal Asymmetric Encryption Padding" (OAEP) under which RSA is claimed to be IND-CCA secure. The importance of OAEP came into effect in 1998, when Bleichenbacher [26] showed a chosen ciphertext attacks against PKCS1 v1.5 based on the RSA Encryption. This attack compelled the research community to change the current PKCS standard and apply the OAEP technique as PKCS1v2.0. OAEP technique of message pre-processing provides a way to create public key encryption scheme with higher level security using weakly secure one-way cryptosystem. With time, this message pre-processing technique in case of asymmetric encryption also known as *asymmetric message padding* or simply *message padding*.

It is well known that asymmetric cryptography remains suitable for exchanging short messages or exchanging key only. A concept of hybrid encryption was introduced by Cramer and Shoup [44]. Hybrid encryption is an asymmetric encryption system but a combination of appropriate asymmetric encryption and symmetric encryption. This hybrid encryption provides presence of a symmetric key in encapsulated manner using asymmetric encryption without pre-establishment of a common secret key between two parties. Hybrid encryption exploits the efficiency of the symmetric cryptosystem to encrypt arbitrary long messages with the encapsulated key. In hybrid encryption, asymmetric part is known as key encapsulation mechanism (KEM) which takes the public key as input parameter and outputs a symmetric key and its encryption as encapsulation. Symmetric part of hybrid encryption is known as data encapsulation mechanism (DEM) which uses symmetric key given by KEM and encrypts the input message into a ciphertext. The final output of the system is key encapsulation and ciphertext. RSA-OAEP played an important role as KEM candidate. OAEP allows RSA-OAEP to be used as KEM as well.

Other than encryption, authentication is also an important aspect of cryptography. As hash functions are a prime base for providing authentication in symmetric key cryptography; similarly, digital signature schemes play an important role in providing authentication in asymmetric cryptography. Digital signature schemes are techniques to assure an entity's acknowledgment of having sent a certain message. Typically, an entity has a private key and a corresponding public key which is tied to the entity's name. The entity generates a string called as a signature which depends on the message to sign and his private key. The fact that the entity acknowledged, i.e. that he signed the message, can be verified by anyone using the entity's public key, the message, and the signature. Data authentication and signature schemes are distinguished in the sense that in the latter, verification can be done by anyone at any time after the generation of the signature. Due to this property, the digital signature scheme achieves non-repudiation property, that is, a signer cannot later deny the fact of signing. As part of security, a prime expectation from a signature scheme is unforgeability, where an adversary can not create a valid signature on a message. Development of signature schemes kept running parallel along with research over asymmetric encryption schemes. OAEP, and other padding schemes motivated by OAEP, also played a major role in constructing secure signature schemes from one-way cryptosystems (RSA, ElGamal).

Signing and encryption both are asymmetric operations which are costly computation. Asymmetric cryptography took a step further by merging both into a single system as signcryption. The aim of signcryption is to provide both confidentiality and authentication of messages more efficiently than performing encryption and signing independently. The reduction of the computational cost makes signcryption more practical, and it is a preferred option for e-commerce and e-mail applications, where both confidentiality and authentication are required. Zheng [103] introduced the notion of signcryption in 1997. OAEP and other different padding motivated by OAEP were found to be useful for having a secure signcryption scheme [52, 53, 88] for achieving confidentiality and authenticity together.

Motivation: It is quite evident that OAEP has its influence across different areas of cryptography which makes OAEP an interesting topic of further research. This interest has brought many research works, for instance [2, 9, 27, 28, 41, 52, 63, 65, 83, 86, 87, 98] in literature. Any improvement related to OAEP will generate a improvement chain in many different areas of asymmetric cryptography. With this motivation in this work, we start our journey with OAEP and its development in efficiency and security over time. We also discuss the development of asymmetric encryption with OAEP. We provide an alternative of OAEP and verify its application. To start with detailed explanations of various design and security, we provide some definitions and system design to understand rest of the chapters.

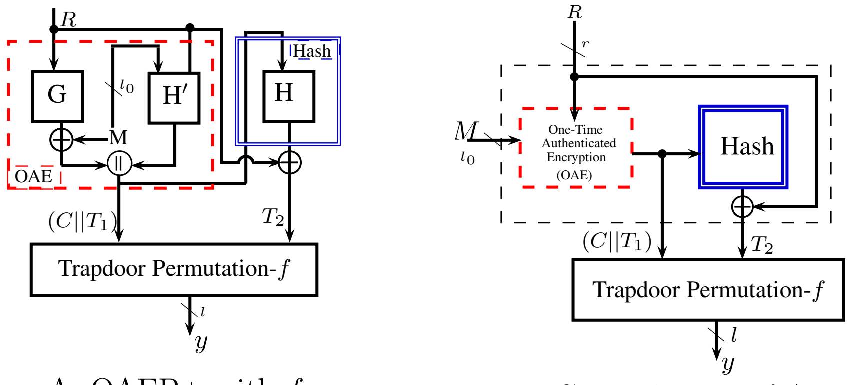
1.2.2 RSA OAEP

In 1994, Bellare and Rogaway proposed a generic conversion [13], in the random oracle model, the "Optimal Asymmetric Encryption Padding" (OAEP), which was claimed to apply to any family of trapdoor one-way permutations, such as RSA. The key generation produces a one-way permutation $f : \{0,1\}^k \rightarrow \{0,1\}^k$, the public key. The private key is the inverse permutation f^{-1}, which requires a trapdoor to be actually computed. The scheme involves two hash functions $G : \{0,1\}^{k_0} \rightarrow \{0,1\}^{n+k_1}$ and $H : \{0,1\}^{n+k_1} \rightarrow \{0,1\}^{k_0}$,where $k = k_0 + k_1 + n + 1$. For any message $m \in \{0,1\}^n$ to be encrypted, instead of computing $f(m)$, as done with the above plain-RSA encryption, one first modifies M. For that, one chooses a random string $R \in \{0,1\}^{k_0}$; computes $C = (M||0^{k_1}) \oplus G(R)$ and $T_1 = R \oplus H(C)$; finally, computes $y = f(C||T_1)$.

The decryption algorithm first computes $P = f^{-1}(y)$, granted the private key, the trapdoor to compute f^{-1}, and parses it as $P = C||T_1$. Then, one can get $R = T_1 \oplus H(C)$, and $M' = C \oplus G(R)$, which is finally parsed into $M' = M||0^{k_1}$, if the k_1 least significant bits are all 0. For a long time, the OAEP conversion has been widely believed to provide an IND-CCA encryption scheme from any trapdoor one-way permutation. However, the sole proven result was the semantic security against non-adaptive chosen-ciphertext attacks (a.k.a. lunchtime attacks [79]). In 2002, Shoup [97, 98] showed that it was very unlikely that a stronger security result could be proven. However, because of the wide belief of a strong security level, RSA-OAEP became the new PKCS #1 v2.0 for encryption after an effective attack against the PKCS #1 v1.5 [26].

Shoup [97, 98] also presents a new scheme called OAEP+, along with a proof of security in the random oracle model. OAEP+ is essentially just as efficient as OAEP, and has a tighter security reduction. It should be stressed that these results do not imply that a particular instantiation of OAEP, such as RSA-OAEP, is insecure. They simply undermine the original justification for its security. In fact, it turns out – essentially by accident, rather than by design - that RSA-OAEP is secure in the random oracle model; however, this fact relies on special algebraic properties of the RSA function, and not on the security of the general OAEP scheme.

These observations were subsequently extended in [56] to RSA-OAEP with arbitrary encryption exponent. Fujisaki et al. [56] provided a complete security proof of IND-CCA-security for OAEP in general, but also for RSA-OAEP in



A: OAEP+ with f; B: General View of A

Figure 1.1: Generalization of OAEP+

particular under the RSA assumption.

1.2.3 Generic View of OAEP+

In this section we provide a *general view*[1] of the OAEP+ with f as the trapdoor one way permutation in an informal way. This helps us to elaborate the basis of the design of our work and its development as per required features. This *general view* is shown in Figure 1.1. It has three parts:

1. *One time Authenticated Encryption (OAE)*: This is a one time authenticated encryption that uses a one time key R and generates an encoded message C and Tag T_1 of message M. Message will be padded to suitable length according to OAE.

2. *Hash*: This is a deterministic hashing algorithm. The concatenation of the outputs of OAE with a one time key R is the input of this hashing algorithm. It outputs T_2.

3. *Trapdoor one way permutation*: This is a trapdoor one way permutation $f : \{0,1\}^\ell \to \{0,1\}^\ell$, which takes the concatenation of the outputs of OAE and *Hash* and produces the final encryption.

Figure 1.1 shows OAEP+ construction with f as the trapdoor one way permutation. G, H' and H are the hash functions used in OAEP+. If we map OAEP+ on our *general view* then the combination of G and H' is OAE while

[1]This informal general view helps in understanding our scheme.

H is the Hash part. G provides a kind of one time pad encryption (OTE) to message M, H' provides hash tag T_1 of M and H produces hash tag T_2 of OTE and tag T_1.

We experience a heavy usage of hash functions in OAEP-type schemes. In 2012, a competition [80] for selecting new hash function as the SHA-3 candidate has been completed. In the past, standard hash functions from the MD-family and the SHA-family were based on same design model using a compression function in iterated mode. In SHA-3 competition many new design techniques came forward, out of which Sponge function gathered most of the attention. In the next section, we take a look at the Sponge function and its versatility.

1.3 Sponge function

Sponge functions were introduced by Guido Bertoni, Joan Daemen, Michael Peeters and Gilles Van Assche in ECRYPT Hash Function Workshop 2007 [20]. A Sponge function can be used as a hash function, but can also generate an infinite bit stream, making it suitable to work as a stream cipher or a pseudo-random bit generator. In this section, we provide a brief description of the Sponge function to the extent necessary for understanding its working and properties. For a complete specification, we refer the interested reader to the original specification [19]. The Sponge function works on a b-bit internal state, divided according to two main parameters r and c, which are called bitrate and capacity, respectively. Initially, the $(r + c)$-bit state is filled with 0s, and the message is split into r-bit blocks. Then, the Sponge function processes the message in two phases.

In the first phase (also called the absorbing phase), the r-bit message blocks are XORed into the state, interleaved with applications of the internal permutation. After all message blocks have been processed, the Sponge function moves to the second phase (also called the squeezing phase). In this phase, the first r bits of the state are returned as part of the output, interleaved with applications of the internal permutation. The squeezing phase is finished after the desired length of the output digest has been produced.

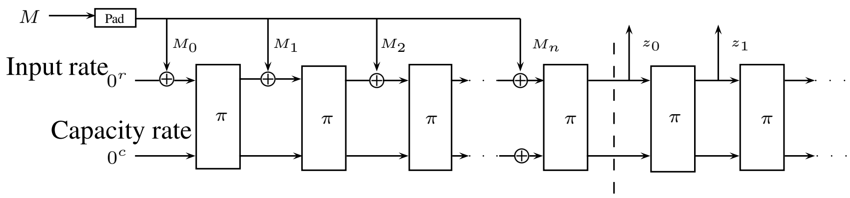


Figure 1.2: Sponge function hash mode

The Sponge function can also be used in keyed mode, providing several different functionalities. A hash-based message authentication code (MAC), a stream cipher and an authenticated encryption (AE) scheme based on the design methods proposed in [21] are some functionalities of Sponge, which we will use in this work.

Due to versatility of Sponge functions, they are quite popular in new designs. Keccak, the SHA-3 winner, is also based on a Sponge function. The popularity of Sponge functions can be seen clearly in CAESAR [18] and PHC [4] competitions.

Performance of Keccak as SHA3 A highly optimized SHA-3 implementation on modern Intel Core CPUs can be executed at a rate of about 13 cycles/byte which translates, e.g., to a throughput of approximately 230 MByte/s (or about 1.84 Gbit/s) if the processor is clocked at 3 GHz. On 8 bit CPUs, which are very popular in embedded systems, SHA-3 can be implemented at about 1110 cycles/byte. Assuming a clock frequency of 10 MHz, this results in a throughput of about 9 kByte/s, or roughly 72 kbit/s.

Keccak turns out to be very well suited for hardware implementations. The algorithm is considerably more efficient in hardware than SHA-2. A high-speed parallelized architecture can easily achieve throughputs of 30 Gbit/sec or beyond with an area of about 100,000 gate equivalences. On the other hand of the performance spectrum, a very small serial hardware engine with less than 10,000 gate equivalences can still achieve throughputs of several 10 Mbit/sec.

Start line of Work "A possible placement of Sponge structure in *general view* described in section 1.2.3". With this thought, we started out journey of using Sponge structure as OAEP-type padding. In OAEP-type and other similar schemes, used H supposed to be replaced by Sponge structure based SHA-3 in practice. Therefore, we focus on showing how we can achieve same and even better properties when we replace H by SHA-3. This improvement could be done only by exploiting structural properties of SHA-3, where SHA-3 structure provides more features than just being a hash function. Therefore, we believe, for efficient use of the base developed for asymmetric encryption message padding over the years, we require to see the things in the different and more granular way. Viewing a hash function H as an open structure (H^π) is a different view where an adversary is more powerful after having access to Sponge permutation π.

1.4 Structure of Thesis

This thesis is based on the incremental motivation of having better version by gathering many features together as a step by step development.

In Chapter 1, we go through some highlights in the development of asymmetric encryption. We observe that pre-processing of messages using message padding (OAEP) plays an important role in development. We showed a *generic view* of these padding schemes. At the end of the chapter, we conclude with a thought of having a new message padding along with a candidate, Sponge structure, to fulfill the *generic view*.

In Chapter 2, we go through some basic definitions and notations related to asymmetric encryption that will be required throughout the work. Begining with this chapter, we start each chapter with some introduction and motivation which build on results and limitations of the previous chapter(s). Throughout we work with asymmetric cryptosystem in a generic way instead of relying on an intractable problem (discrete log problem, integer factorization, etc.).

In Chapter 3, we work on the thought, apply Sponge structure in *generic view*, with which we finished in Chapter 1. We provide detailed description of Sponge based asymmetric encryption padding (SpAEP). We also provide the security proof of this scheme and a comparison with previous schemes. We show that SpAEP performs better than previous proposals. Our proposed padding works with any trapdoor one-way permutations and handles arbitrarily long messages like hybrid encryption but as a monolithic system. At the end of the chapter, we go through the subsequent scope of the work in term of its applicability in hybrid encryption and removing some of its limitations such as decryption overhead and compatibility with trapdoor one-way permutation only.

Portions of this chapter have appeared in the following publication.

- Tarun Kumar Bansal, Donghoon Chang, and Somitra Kumar Sanadhya. *Sponge based CCA2 secure asymmetric encryption for arbitrary length message.* Information Security and Privacy - 20th Australasian Conference, ACISP 2015, Brisbane, QLD, Australia, 2015, Springer, 2015.

In Chapter 4, we propose another version SpRKEM of the scheme SpAEP which provides an extra capability to support hybrid encryption as a two-fold system of asymmetric encryption and symmetric encryption. We also provide the security proof of the scheme and a comparison with previous similar hybrid encryption schemes. This SpRKEM version also helps us in finding a way to solve a decryption overhead limitation of SpAEP.

Portions of this chapter have appeared in the following publication.

- Tarun Kumar Bansal, Donghoon Chang, and Somitra Kumar Sanadhya. *Sponge based CCA2 secure asymmetric encryption for arbitrary length message (extended abstract).* International Journal of Applied Cryptography (IJACT), 2017. Editor in Chief: Dr. Yi Mu, Dr. David Pointcheval . (under printing)

In Chapter 5, we target to apply some modifications in SpAEP to have another Sponge based padding (SpPad). This SpPad is compatible with any one-way secure cryptosystem (Pe) which includes deterministic one-way cryptosystem (e.g. RSA) as well as probabilistic one-way cryptosystem (e.g. ElGamal). In SpPad, we are also able to achieve lower decryption overhead compared to SpAEP. This lower decryption overhead in SpPad–Pe also help in enabling streaming option. We also provide security proof of SpPad–Pe. Comparison of SpPad–Pe with previous schemes, that are also secure with any one-way secure asymmetric cryptosystem, shows SpPad–Pe as better scheme.

At the end of this chapter, we conclude that the *generic view* with few modifications, when filled with Sponge structure works securely and efficiently. In conclusion, we also come up with the question about security of the *generic view* with any generic structure instead of specific Sponge structure as basic underlying primitive. We answer this question in next chapter.

In Chapter 6, we modify the *generic view* as per required modifications we learn through SpPad–Pe. We achieve a generic asymmetric encryption framework, which has all the proposed properties and security in Sponge based message padding cryptosystem. Now we have a generic framework along with security proof in random oracle model, which can be instantiated not only by Sponge but also by other available structures if needed. We call this framework as "Real-time CCA-secure Encryption of Arbitrary Long messages"(REAL). REAL performs well when compared to previous similar works such as FO-transform [54, 55]. At the end of the chapter, we propose a possibility of applying Sponge padding technique in the area of signcryption. We explore more about this possibility in next chapter.

In Chapter 7, we propose a modified version of SpPad which makes it compatible with weakly secure asymmetric encryption and signature primitives to yield a generic signcryption scheme. In signcryption, both encryption and signature are required simultaneously. We provide security proofs for both confidentiality and unforgeability of proposed signcryption scheme. We show that the proposed scheme performs better than generic schemes of previous works. Proposed scheme is the first signcryption scheme based on Sponge structure and offers maximum security using weak underlying asymmetric primitives along with the ability to handle long messages. We also show that the probabilistic and deterministic nature of underlying asymmetric primitives play a crucial role in security of signcryption scheme. With this chapter, we conclude our journey of designing generic asymmetric key cryptosystem using message padding.

Chapter 8 provides final summary and conclusion for the topics covered in this thesis. We propose some directions that could be used as a part of future work.

Chapter 2

Preliminaries

Contents

Notations: In this work, we represent $k \in \mathbb{N}$ as security parameter, where $\mathbb{N}$ is the set of natural numbers. We will use the symbol $|x|$ to denote the bit length of a string x and $x||y$ to denote the concatenation of strings x and y. If n is a positive integer then the symbol $\{0,1\}^n$ denotes the set of n-bit strings. We also use symbol $\{0,1\}^*$ to denote the set of binary strings with no fixed length. $\lfloor x \rfloor_r$ represents first r-bit string of x where $|x| \geq r$. Selecting a uniformly and independently distributed variable x from a set X is denoted by $x \xleftarrow{\$} X$.

Negligibility: A function $negl()$ is negligible if for every polynomial $p(n)$ there exists an $N \in \mathbb{N}$ such that for all integers $n > N$ it holds that $negl(n) < \frac{1}{p(n)}$.

For convenience we will use ε to denote negligible functions.

Random Oracle Model: Hash functions play an important role in constructing any cryptographic scheme. For security proofs a hash function is considered as an ideal randomised black box function, also called random oracle [12]. In brief, random oracle is an ideal random function that is publicly available for computation without knowing its internal structure. For each new arbitrarily long or fixed length input, random oracle outputs a fixed length random output from a fixed range. Security proof considering hash functions as random oracle is denoted as security proof in random oracle model [74].

H is said to be a random oracle from a set X to set Y if for each $x \in X$ the value of *H(x)* is chosen randomly from Y. More precisely, $\Pr[H(x){=}y | H(x_1) = y_1, H(x_2) = y_2, ...H(x_q) = y_q] = \frac{1}{M}$, where $x \notin \{x_1, x_2, \ldots x_q\}$, $y, y_1, \ldots, y_q \in Y$, $|Y| = M$ and q is the total number of queries. If H accepts variable length input it is considered as VIL Random Oracle.

Ideal permutation: A permutation π is a bijective function on a finite domain D and finite range R with $D = R$. An ideal permutation is a permutation chosen uniformly at random from all the available permutations. Let $D = R = \{0,1\}^b$, then $\pi \xleftarrow{\$} \mathrm{Perm}(D, D)$, where $\mathrm{Perm}(D, D)$ is the collection of all permutations on D. Mathematically, $\pi : D \to R$ is a permutation, if for every $y \in R$ there is one and only one $x \in D$ such that $\pi(x) = y$.

2.1 Trapdoor One-way functions

Definition 1. ***Family of functions***. *A family of functions $\mathcal{F}$ =\{Gen, SampI, SampR, Eval\} is a tuple of four algorithms which works as follows:*

- *The randomised key generation algorithm "Gen" takes a security parameter $k \in \mathbb{N}$ and outputs a pair (pk,sk) where pk is public key and sk is related private key.*

- *The randomised sampling algorithm "SampI" takes input pk and returns a random value x in a set that we call the domain of pk and denoted by $Dom_{\mathcal{F}}(\mathsf{pk})$ and with $|x| \geq k$*

- *The randomised sampling algorithm "SampR" takes input* pk *and returns a random value* g *in a set that we call the randomness domain of* pk *and denoted by* $COIN_{\mathcal{F}}(\mathsf{pk})$ *and with* $|g| \geq k$

- *Evaluation algorithm "Eval" takes input* pk *a point* $x \in Dom_{\mathcal{F}}(\mathsf{pk})$ *and* $g \in COIN_{\mathcal{F}}(\mathsf{pk})$. *Eval returns an output we denote by* $Eval_{pk}(x; g)$. *We denote range of function* $Eval_{pk}(x; g)$ *by* $Rang_{\mathcal{F}}(pk)$, *i.e.* $Rang_{\mathcal{F}}(\mathsf{pk}) = \{Eval_{\mathsf{pk}}(x; g) | x \in Dom_{\mathcal{F}}(\mathsf{pk}), g \in COIN_{\mathcal{F}}(\mathsf{pk})\}$

We say $\mathcal{F}$ *is a family of permutations if for all values of* pk $COIN_{\mathcal{F}}(\mathsf{pk}) = \emptyset$, $Dom_{\mathcal{F}}(\mathsf{pk}) = Rang_{\mathcal{F}}(\mathsf{pk})$ *and* $Eval_{\mathsf{pk}}()$ *is a permutation on set* $Dom_{\mathcal{F}}(\mathsf{pk})$.

Definition 2. Families of Trapdoor functions. $\mathcal{F}$ *is a family of trapdoor functions if there exists a deterministic inversion algorithm "Inv" that takes input* sk *and a point* $y \in Rang_{\mathcal{F}}(pk)$ *and return a point* $x \in Dom_{\mathcal{F}}(\mathsf{pk})$ *such that* $Eval_{\mathsf{pk}}(x; g) = y$ *for all* $g \in COIN_{\mathcal{F}}(\mathsf{pk})$.

We say $\mathcal{F}$ *is a family of trapdoor permutations if for all values of* pk $COIN_{\mathcal{F}}(\mathsf{pk}) = \emptyset$, $Dom_{\mathcal{F}}(\mathsf{pk}) = Rang_{\mathcal{F}}(\mathsf{pk})$ *and* $Eval_{\mathsf{pk}}()$ *is a permutation on set* $Dom_{\mathcal{F}}(\mathsf{pk})$.

We describe definition of one-wayness.

Definition 3. **(θ-one-way)**. *Let* $\mathcal{F} = \{Gen, SampI, SampR, Eval\}$ *be a family of trapdoor functions. Let* $d \in \{0, 1\}$, $k \in \mathbb{N}$ *be a security parameter. Let* $\mathcal{A}$ *be an adversary and* $0 < \theta \leq 1$ *be a constant. Consider the following experiment:*

$Exp^{\theta-one-way}_{\mathcal{F},\mathcal{A}}(k)$

1. $(\mathsf{pk}, \mathsf{sk}) \xleftarrow{\$} Gen(k)$; $g \xleftarrow{\$} COIN_{\mathcal{F}}(\mathsf{pk})$.
2. $x_1 || x_2 \xleftarrow{\$} Dom_{\mathcal{F}}(\mathsf{pk})$, *where* $|x_1| = \lceil \theta \cdot |(x_1||x_2)| \rceil$
3. $y \leftarrow Eval_{\mathsf{pk}}(x_1||x_2)$
4. $x_1^{'} \leftarrow \mathcal{A}(\mathsf{pk}, y)$, *where* $|x_1^{'}| \geq |x_1|$
5. **if** $x_1^{'} == x_1$ **then**
   Return 1
   else
   Return 0

We define the advantage of $\mathcal{A}$ *as*

$$Adv^{\theta-ow}_{\mathcal{F},\mathcal{A}}(k) = \Pr[Exp^{\theta-one-way}_{\mathcal{F},\mathcal{A}}(k) = 1]$$

We consider that the family $\mathcal{F}$ is θ-one-way if $Adv^{\theta-ow}_{\mathcal{F},\mathcal{A}}(k)$ is negligible for any adversary $\mathcal{A}$ whose time complexity is polynomial in k. If $\theta = 1$ then we consider $\mathcal{F}$ is family of one-way functions over entire input excluding random coins.

Asymmetric one-way cryptosystem Starting with $\mathcal{F}$, an asymmetric one-way cryptosystem Pe: $(Gen^{\mathcal{F}}, \mathsf{Enc}^{\mathcal{F}}, \mathsf{Dec}^{\mathcal{F}})$ is obtained in the following way: the keys $(\mathsf{pk}, \mathsf{sk}) \xleftarrow{\$} Gen(k)$, the ciphertext for message(plaintext) $x \in Dom_{\mathcal{F}}(pk)$ with randomness $g \xleftarrow{\$} COIN_{\mathcal{F}}(\mathsf{pk})$ is $y = \mathsf{Enc}^{\mathcal{F}}(\mathsf{pk}, x; g) = Eval_{\mathsf{pk}}(x; g)$ and a valid ciphertext $y \in Rang_{\mathcal{F}}(pk)$ is decrypted by means of $\mathsf{Dec}^{\mathcal{F}}(\mathsf{sk}, y) = Inv_{\mathsf{sk}}(y) = x$.

If an asymmetric cryptosystem follows the minimum security requirement of one-wayness then we treat that cryptosystem as asymmetric primitive Pe. If Pe does not uses random coins g, we say Pe is a deterministic asymmetric encryption otherwise it is called a probabilistic asymmetric encryption.

We denote ℓ as bit length of $x \in Dom_{\mathcal{F}}(\mathsf{pk})$, $\ell + co_{pe}$ as bit length of $y \in Rang_{\mathcal{F}}(\mathsf{pk})$ and λ is length of $g \in \mathtt{COINS}$, where $\lambda \geq k$

In case of family of trapdoor one-way permutations $\mathcal{F}$, we represent permutation $Eval_{pk}(\cdot)$ as function $f(\cdot)$ and $Inv_{\mathsf{sk}}(\cdot)$ as inverse function $f^{-1}(\cdot)$.

2.2 Public-Key Encryption

Description: A public-key encryption scheme Encrypt is defined by three algorithms:

- The key generation algorithm $\mathsf{GenEnc}(1^k)$ produces a pair $(\mathsf{pk}, \mathsf{sk})$ of public and private keys on input 1^k, where k is the security parameter.

- The encryption algorithm $\mathsf{Enc}_{\mathsf{pk}}(m; g) = c$ outputs a ciphertext c for input a message $m \in \mathbb{M}$ and pk using random coins $g \in \mathtt{COINS}$. The message and coin spaces, $\mathbb{M}$ and $\mathtt{COINS}$, are uniquely determined by pk.

- The decryption algorithm $\mathsf{Dec}_{\mathsf{sk}}(c)$ outputs the associated message m.

We require that an asymmetric encryption scheme should satisfy the following correctness condition: For every sufficiently large $k \in \mathbb{N}$, for all $(\mathsf{pk}, \mathsf{sk})$ generated by $\mathsf{GenEnc}(1^k)$, and every $m \in \mathbb{M}$ and $g \in \mathtt{COINS}$, we always have $\mathsf{Dec}_{\mathsf{sk}}(\mathsf{Enc}_{\mathsf{pk}}(m, g)) = m$.

Security Notion: The simplest security notion for a public key encryption, say $\textsc{Encrypt}$, is one-wayness (OW): with public data only, an adversary $\mathcal{A}$ cannot recover the whole plaintext m of a given ciphertext c. We denote by $\mathsf{Succ}^{\mathsf{OW}}_{\mathcal{A},\textsc{Encrypt}}$ the maximum probability of success that $\mathcal{A}$ can invert the encryption of a random plaintext m. OW is minimal security requirement for any asymmetric cryptosystem. We consider such asymmetric cryptosystem, which follows OW, as asymmetric primitive.

A variant of one-wayness is $\mathsf{OW\text{-}PCA}$, introduced in [83], because of probabilistic asymmetric one-way encryption schemes. This notion of one-wayness is considered when adversary has access to a *Plaintext checking oracle* ($\mathcal{O}^{PC}$). The goal of $\mathcal{A}$ is the same as for OW but she is given access to a plaintext-checking oracle ($\mathcal{O}^{PC}$) along with other public information. $\mathcal{O}^{PC}$ outputs 1 if a given (m, c) pair is a valid message-ciphertext pair for $\textsc{Encrypt}$, otherwise it returns 0. As shown in [83], the ElGamal [57] encryption achieves $\mathsf{OW\text{-}PCA}$ under GDH assumption [82]. Evidently, for asymmetric encryption scheme based on trapdoor one-way permutation, the notion of OW and $\mathsf{OW\text{-}PCA}$ are the same.

A stronger security notion has also been defined. It is the so-called semantic security (a.k.a. indistinguishability of encryptions, IND) [58]. If an attacker has some information about the plaintext, the view of the ciphertext should not leak any additional information. This security notion more formally considers the advantage an adversary can gain when trying to guess, between two messages, which one has been encrypted. In other words, an adversary is seen as a 2-stage Turing machine $(\mathcal{A}_1, \mathcal{A}_2)$, and the advantage $\mathsf{Adv}^{\mathsf{ind}}_{\textsc{Encrypt}}(\mathcal{A})$ should be negligible for any adversary, where

$$Adv^{\mathsf{ind}}_{\textsc{Encrypt}}(\mathcal{A}) = 2 \times \Pr\left[\begin{array}{l}(\mathsf{pk},\mathsf{sk}) \leftarrow \mathsf{GenEnc}(1^k), (m_0, m_1, s) \leftarrow \mathcal{A}_1(\mathsf{pk}),\\ b \in \{0,1\}, c = \mathsf{Enc}_{\mathsf{pk}}(m_b) : \mathcal{A}_2(m_0, m_1, s, c) = b\end{array}\right] - 1$$

On the other hand, an attacker can use many kinds of attacks, depending on the information available to him. First, in the public-key setting, the adversary can encrypt any plaintext of his choice with the public key: this basic scenario is called the chosen-plaintext attack, and denoted by CPA. Extended scenarios allow the adversary a restricted or unrestricted access to various oracles. The main and strongest one is the decryption oracle which can be accessed adaptively in the chosen-ciphertext scenario, denoted CCA. There is the natural restriction that any query to this oracle should be different from the challenge ciphertext. A general study of these security notions and attacks was conducted in [11, 74, 92, 93].

In this work, we denote an asymmetric primitive by Pe. We would like to remind that we consider a asymmetric cryptosystem which only follows OW as asymmetric primitive.

2.3 Signature Schemes

Description A digital signature scheme SIGN consist of three algorithms:

- $\mathsf{GenSign}$, the key generation algorithm which for security parameter k, on input 1^k, outputs a pair (pk,sk) of matching public and private keys;
- Sign, the signing algorithm which receives a message M and the private key sk, and outputs a signature $\sigma = \mathsf{Sign}_{\mathsf{sk}}(M)$;
- Ver, the verification algorithm which receives a candidate signature σ, message M, and a public key pk, and returns an answer $\mathsf{Ver}_{\mathsf{pk}}(\sigma, M)$ as to whether σ is a valid ($\top$) or invalid ($\perp$) signature of M with respect to pk.

We suppose signing algorithm take input of maximum ℓ_{sg} bits and that output length of signing algorithm is ℓ_σ .

Security notions. The attacker attempts to forge a signature. The probability of achieving this is assessed via the following game between a probabilistic, polynomial-time (PPT) attacker and a hypothetical challenger:

1. The challenger generates a key pair $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{GenSign}(1^k)$.
2. The attacker runs $\mathcal{A}^{\mathcal{O}}(1^k, \mathsf{pk})$. The attacker has access to an oracle $\mathcal{O}$ (which will be described subsequently). The attacker terminates by outputting a message m^* and a signature σ^*.

In terms of resources, there are two types of attacks. The type of attack specifies the power that the attacker has in the attack.

- In a *no-message attack* (NMA), the oracle gives no response. This is equivalent to an attack model in which the attacker does not have access to the oracle $\mathcal{O}$. The attacker only knows public key pk of the signer.

- In second, *known-message attacks*, the attacker has access to a signature oracle providing list of valid message/signature pairs in addition to knowledge of public key of the signer. If this list contains random and uniformly chosen messages, then the attack is termed as "random-message attack (RMA)". If this list contains messages chosen by adversary, the the attack is termed as "chosen-message attack (CMA)". A chosen message attack seeks to emulate the normal mode of use of a signature scheme, in which an attacker can observe signatures produced by a legitimate party, perhaps in some adversarial chosen way. Therefore, in adaptive chosen message attack (Ada) adversary chose messages in adaptive way.

There are two ways in which we can assess whether the attacker succeeds in forging a signature.

- In the existential unforgeability (UF) game, the attacker is said to win if it outputs a pair (m^*, σ^*) where $\mathsf{Ver}_{\mathsf{pk}}(m^*, \sigma^*) = \top$ and the attacker never queried the signature oracle with the message m^*.
- A slightly stronger notion of security is that of strong existential unforgeability(sUF). The attacker is said to win the strong unforgeability game if it outputs a pair (m^*, σ^*) where $\mathsf{Ver}_{\mathsf{pk}}(m^*, \sigma^*) = \top$ and the attacker never queried the signature oracle with the message m^* and received the response σ^*.

In case of finite message space $\mathcal{M}$, we may consider weaker security notion. For success criteria, we may ask the attacker to produce a forged signature for a randomly chosen message $m^* \xleftarrow{\$} \mathcal{M}$. This leads to a new description for the attack game that a probabilistic, polynomial-time attacker $\mathcal{A}$ is playing:

1. The challenger generates a key pair $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{GenSign}(1^k)$ and a message $m^* \xleftarrow{\$} \mathcal{M}$.
2. The attacker runs $\mathcal{A}^{\mathcal{O}}(1^k, \mathsf{pk}, m^*)$. The attacker has access to an oracle $\mathcal{O}$. The attacker terminates by outputting a signature s^*.

Again, we may define two success criteria for this security game:

- In the universal unforgeability (uUF) game, the attacker is said to win if $\mathsf{Ver}_{\mathsf{pk}}(m^*, \sigma^*) = \top$ and the attacker never queried the signature oracle with the message m^*.

- In the strong universally unforgeability (suUF) game, the attacker is said to win if $\mathsf{Ver}_{\mathsf{pk}}(m^*, \sigma^*) = \top$ and the attacker never queried the signature oracle with the message m^* and received the response σ^*.

2.4 Hybrid Encryption

2.4.1 Key Encapsulation Mechanism: KEM

Description: (KEM). A key encapsulation mechanism is defined by KEM = (KEM.Gen, KEM.Encap, KEM.Decap) as an ordered tuple of three algorithms.

1. A probabilistic key generation algorithm KEM .Gen. It takes as input a security parameter k, and outputs a private/public keypair $(\mathsf{sk}, \mathsf{pk})$. As part of the public key there is a parameter KEM.keylen that specifies the length of the symmetric keys used by symmetric cipher.

2. A probabilistic key encapsulation algorithm PKE.Encap. It takes as input a public key pk, and outputs a symmetric key K of length KEM.keylen, and an encapsulation ψ.

3. A deterministic decapsulation algorithm PKE.Decap. It takes as input a private key sk and an encapsulation ψ and outputs either a key K or the unique error symbol $\perp$.

The KEM is sound if for almost all valid keypairs $(\mathsf{sk}, \mathsf{pk})$, whenever (K, ψ) was the output of PKE.Encap(pk), we have K = PKE.Decap(sk, ψ).

2.4.2 Data Encapsulation Mechanism: DEM

A data encapsulation mechanism (DEM) is used to encrypt long message (or part of message) using symmetric key K generated by the KEM. The DEM is more like a symmetric encryption scheme with a different key during each encryption. There are two security notions for DEM. The first one is message indistinguishability and the second one is the ciphertext integrity.

Description: Data encapsulation mechanism DEM = (DEM.Enc, DEM.Dec) consist of a pair of two algorithms which are described next:

1. Encryption algorithm DEM.Enc takes a message M and a symmetric key K of length DEM.keylen for the security parameter k, and outputs a ciphertext $\chi = C^e||Tag$.
2. Decryption algorithm DEM.Dec takes a ciphertext $\chi = C^e||Tag$ and symmetric key K for the security parameter k, and outputs either a message M or $\perp$.

Definition 4. ***IND-PA/INT-CTXT game for DEM***: *A challenger and an adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ play a IND-PA/INT-CTXT game for a given DEM. Having a security parameter k, the game runs as follows.*

Experiment: $Exp^{IND-PA}_{DEM,\mathcal{A}}(k)$

1. $K \xleftarrow{\$} \{0,1\}^k$
2. $(M_0, M_1, s) \leftarrow \mathcal{A}_1(1^k)$
3. $d \xleftarrow{\$} \{0,1\};$
4. $(\chi^*) = DEM.Enc(M_d, K)$
5. $d' \leftarrow \mathcal{A}_2(\chi^*, s)$
6. **return** d'

$\mathcal{A}$ wins the game if $d = d'$. The advantage of $\mathcal{A}$ is given as

$$Adv^{IND-PA}_{DEM}(\mathcal{A}) =\mid Pr[\mathcal{A}\ wins] - \tfrac{1}{2} \mid$$

Experiment: $Exp^{INT-CTXT}_{DEM,\mathcal{A}}(k)$

1. $K \xleftarrow{\$} \{0,1\}^k$
2. $(M, s) \leftarrow \mathcal{A}_1^{DEM.Dec(\cdot)}$
3. $(\chi^*) = DEM.Enc(M, K)$
4. $\chi' \leftarrow \mathcal{A}_2^{DEM.Dec(\cdot)}(\chi^*, s)$

$\mathcal{A}$ wins the game if $\chi = \chi'$ is a valid ciphertext. The advantage of $\mathcal{A}$ is given as

$$Adv^{INT-CTXT}_{DEM}(\mathcal{A}) = Pr[\mathcal{A}\ wins]$$

2.4.3 (KEM+DEM) Construction

Given a KEM and a DEM, where the keys output by the KEM are of correct length for use with the DEM, i.e. DEM.keylen = KEM.keylen, we construct a hybrid PKE scheme as follows:

- The key generation algorithm PKE.Gen is implemented using KEM.Gen.
- The encryption algorithm PKE.Enc is implemented as follows.
    1. Compute a key/encapsulation pair $(K, \psi) = \text{KEM.Encap}(\mathsf{pk})$.
    2. Encrypt the message to obtain a ciphertext $\chi = \text{DEM.Enc}_K(m)$.
    3. Output the ciphertext $c = (\psi, \chi)$.
- The decryption algorithm PKE.Dec is implemented as follows.

1. Parse the ciphertext to obtain $(\psi, \chi) = c$.
2. Compute the symmetric key $K =$ KEM .Decap(sk, ψ).
3. If $K =\perp$, return $\perp$ and Halt.
4. Decrypt the message $m =$ DEM .Dec$_K(\chi)$.
5. If $m =\perp$, return $\perp$ and Halt, else output m

2.5 Sigcryption: Joint Encryption and Signing

Description . A signcryption scheme SignCrypt is defined by three algorithms:

- Gen, the key generation algorithm which outputs a pair of keys (SDK,VEK) for a security parameter k. SDK is the user's sign/decrypt key, which is kept secret, and VEK is the user's verify/encrypt key, which is made public.
- SignEnc, the encryption and signing algorithm which, for a message M, the public key of the receiver VEK_R and private key of sender SDK_S, produce a signed ciphertext $Y = \mathsf{SignEnc}_{\mathsf{SDK}_S,\mathsf{VEK}_R}(M)$
- VerDec, the decryption and verifying algorithm which, for signed-ciphertext Y, the private key SDK_R of the receiver and the public key VEK_S of the sender, recovers the message $M = \mathsf{VerDec}_{\mathsf{SDK}_R,\mathsf{VEK}_S}(Y)$. If this algorithm fails either to recover the message or to verify its authenticity, it returns $\perp$.

Security Notions . We can combine classical security notions of signature and encryption to form security notion of signcryption, under adaptive attacks. Given access to public information, PUB=$(\mathsf{VEK}_S, \mathsf{VEK}_R)$, and oracle access to the functionalities of both sender S and receiver R, the adversary attempts to break:

1. authenticity (UF): come up with a valid signed-ciphertext of a new message, and thus provide an "existential forgery".
2. privacy (IND): breaks the "indistinguishability" of signed-ciphertexts.

In the security considerations the adversary may be one of S or R themselves. So, S may want to break the privacy, or R may want to break authenticity. If a signcryption scheme prevents existential forgeries and guarantees indistinguishability, in attack scenarios shown in section 2.3 and section 2.2, called adaptive attacks(AdA and CCA), we say the scheme is secure.

Definition 5. *A signcryption scheme is* ***secure*** *if it achieves* $\mathsf{IND/UF}$ *under adaptive attacks.*

2.6 SpongeWrap and Sponge Function

Bertoni et al. [20–22] proposed the SpongeWrap and Sponge function which are based on an iterated permutation $\pi : \{0,1\}^{(b=r+c)} \to \{0,1\}^b$ with an initial value IV. Because iterated permutation works on fixed length block size, it requires an injective reversible padding. This padding-unpadding function is defined and customized as per requirement of system. A pseudo-code of SpongeWrap and Sponge function is provided in figure 2.1. Both functions uses permutation π of $b = r + c$-bit input, where r is called input rate and c is called capacity rate. SpongeWrap works in both forward ($SpongeWrap^+$) and inverse ($SpongeWrap^-$) direction therefore widely used as encryption or authenticated encryption. Sponge function works in only forward direction, therefore preferred as hash function.

The $pad - unpad$ function of Sponge structure can be defined in various ways. We have taken a generic pad function uses 10*1 injective-reversible padding; this takes two inputs, one as input to be padded, second as input rate of π. One more optional input to the pad function can be added as minimum output length $\ell = n \cdot r$ required from pad depending upon system requirement, where $n \in \mathbb{N}$ and $n \geq 1$. $unpad$ is defined as inverse process of pad.

$\underline{pad(x, r, \ell)}$

$$= \begin{cases} x||1||0^{(\ell-|x|-2)}||1, & \text{if} |x| \leq (\ell - 2) \\ x||1||0^{(r-1)}||1, & \text{if } |x| = \ell - 1 \\ x||1||0^{r-((|x|+1) \bmod r)-1}||1, & \text{otherwise} \end{cases}$$

$\underline{unpad(y, r, \ell)}$

if $\exists\, x \neq \emptyset$ *s.t.* $y = x||1||0^z||1$ *where* $0 \leq z \leq \ell - 3$ *if* $|x| = \ell$ *or* $0 \leq z \leq r - 1$ *if* $|x| > \ell$
then
  return x
else
  return $\perp$

For security parameter k, a permutation π with parameters r and c can be chosen as explained in [20–22]. In abstract, following relation $r \geq c \geq 2k$ is

$\underline{SpongeWrap^{+}(K,M)}$

1. $x||w = IV$, where $|x| = r$
2. $m_1||m_2||\ldots||m_n = pad(M, r, \ell)$, where $|m_i| = r\ \forall 1 \le i \le n$.
3. $x = x \oplus K||0^{r-k}$
4. **for** $i = 1 \to n$ **do**
   - $x||w = \pi(x||w)$
   - $x = x \oplus m_i$
   - $c_i = x$
5. $x||w = \pi(x||w)$; $T = \lfloor x \rfloor_k$
6. Return $c_i||\ldots||c_n||T$

$\underline{SpongeWrap^{-}(K,C||T)}$

1. $x||w = IV$, where $|x| = r$
2. $c_1||c_2||\ldots||c_n = C$, where $|c_i| = r\ \forall 1 \le i \le n$.
3. $x = x \oplus K||0^{r-k}$
4. **for** $i = 1 \to n$ **do**
   - $x||w = \pi(x||w)$
   - $m_i = x \oplus c_i$
   - $x = c_i$
5. $x||w = \pi(x||w)$; $T' = \lfloor x \rfloor_k$
6. **if** T=T' **then**
   - Return $unpad(m_1||\ldots||m_n)$

   else
   - Return $\perp$

$\underline{Sponge(J)}$

1. $x||w = IV$, where $|x| = r$
2. $j_1||\ldots||j_n = pad(J, r, \varnothing)$, where $|j_i| = r\ \forall 1 \le i \le n$.
3. **for** $i = 1 \to n$ **do**
   - $x||w = \pi(x||w)$
   - $x = x \oplus j_i$
4. $x||w = \pi(x||w)$
5. Return $\lfloor x \rfloor_k$

Figure 2.1: Pseudo-code of SpongeWrap and Sponge function

followed.

A combined graphical representation of SpongeWrap and Sponge function is shown in Figure 2.2.

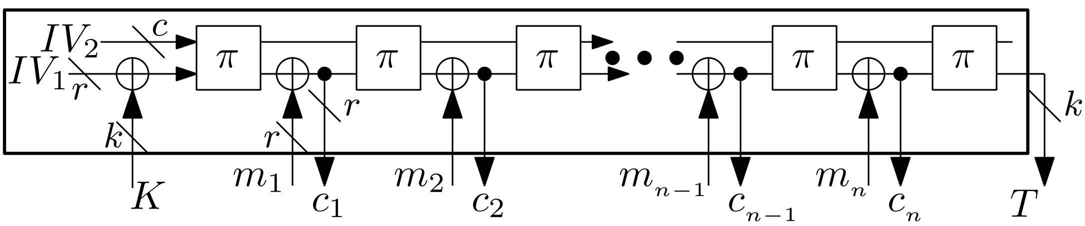


Figure 2.2: 1. **SpongeWrap** : $\{SpongeWrap^{+}, SpongeWrap^{-}\}$ is based on iterated permutation π. By default, Initial Value (IV) is considered as 0 (IV=0^b). During encryption $SpongeWrap^{+}$: On input message ($M = m_1||\ldots||m_n$) and a random K, and output $C||T$ where $C = c_1||\ldots||c_n$, $|c_i| = r\ \forall\ 1 \ge i \ge n$ and, if authentication also required then, $|T| = k$. During Decryption $SpongeWrap^{-}1$: takes $(K, C||T)$ as input and outputs either M or $\perp$.
2. **Sponge** function: Shown figure can be viewed as Sponge function by considering input $J = K||m_1||\ldots||m_e$, by replacing IV_2 from IV_3 and considering T as only output.

We will see more about Sponge structure and its usage in this work. A glimpse of power and importance of Sponge structure can be seen from the fact that Sponge based hash function Keccak [23] has been selected as the winner of the

SHA-3 competition [80], and the popularity of Sponge functions can also be seen in CAESAR [18] and PHC [4] competitions.

Chapter 3

Sponge based CCA secure Asymmetric Encryption from trapdoor one-way permutations

Contents

In this chapter, we introduce a *Sponge based Asymmetric Encryption Padding* scheme (SpAEP), a novel way to use the SpongeWrap [21] and the Sponge function [20] to encrypt arbitrary length messages in Asymmetric key cryptography.

In upcoming sections, first, we discuss OAEP and its development in different versions. We further discuss the motivation of constructing SpAEP as a better alternative to OAEP type schemes. We describe how construction of SpAEP is derived from *generic view* of OAEP+ discussed in section 1.2.3. We provide detailed features and comparison of SpAEP with other OAEP-type schemes as part of contribution. Security proof of proposed scheme is followed by detailed description of SpAEP. This chapter ends with a conclusion and we also look at some of its limitations.

3.1 Background

3.1.1 Different versions of OAEP

OAEP+ was proposed in 2001 by Shoup [97, 98]. OAEP+ uses three "ideal" hash functions of which only two functions can run in parallel. This makes OAEP+ a two pass scheme. The original OAEP construction was also two pass but the construction was based on two "ideal" hash functions. OAEP+ is essentially just as efficient as OAEP, and even has a tighter security reduction. Security proof of OAEP+ is valid with any trapdoor one-way permutation, whereas OAEP security proof is valid with some specific trapdoor one-way permutations.

In 2001, Boneh [29] proposed two much simpler padding schemes than OAEP/OAEP+ for the RSA and Rabin trapdoor permutations that can be CCA secure in the random oracle model. The first one is called Simple-OAEP, or SAEP, and it is based on one "ideal" hash function while the second one is called SAEP+ and it is based on two "ideal" hash functions. Both SAEP and SAEP+ are single pass schemes. The main limitation of SAEP/SAEP+ is that they restrict the message size and their security proof is valid only for the RSA and the Rabin functions. Under similar restrictions, another scheme ZAEP is introduced in [9] which aims to lower the ciphertext overhead by reducing the redundancy in the scheme.

In 2003, Phan and Pointcheval [86, 87] introduced OAEP-3R which is RCCA secure ("relaxed CCA" [87] equivalent to "replayable CCA" [35]- a slightly weaker notion than general CCA) with any trapdoor one way permutation (f) in random oracle model (ROM). Let "ciphertext overhead" [2] stand for the difference between the length of ciphertext and plaintext. OAEP-3R was shown to have only t-bit ciphertext overhead, whereas OAEP and OAEP+ have $3t$-bit ciphertext overhead, where t stands for security requirement in bits[1].

[1]A security requirement of t-bit implies that at-least 2^t queries are required to break the

In 2008, Abe, Kiltz and Okamoto [2] showed that security reduction of OAEP-3R forces ciphertext overhead to be $2t$. A new scheme called OAEP-4X was introduced in [2] which provides CCA security for any trapdoor one way permutation in ROM. OAEP-4X has only t-bit ciphertext overhead which was shown to be optimal (lowest achievable bound). In OAEP-4X, reduction of t-bit ciphertext overhead with respect to OAEP-3R has only limited practical application such as in a highly bandwidth constrained network. Therefore, for general applications ciphertext overhead reduction by t bits is a less interesting case.

The number of hash functions used in OAEP is two and these are used in a 2 round structure. OAEP+ is also 2 round structure but uses three hash functions (two hash function can run in parallel while encryption). OAEP-3R is 3 round structure that uses three hash functions and OAEP-4X is 4 round structure that uses four hash functions. Each of these schemes (OAEP, OAEP+, OAEP-3R and OAEP-4X), proven secure in ROM, requires one or more hash functions with arbitrary size output. For example, for RSA-2048 (or RSA-3072) trapdoor one-way permutation, minimum number of hash function with arbitrary size output required in OAEP, OAEP+, OAEP-3R and OAEP-4X are 1, 1, 2 and 2 respectively.

Currently, no cryptographic standard specifies an instantiation for a hash function of arbitrary size. However, some such instantiations are implicitly required in PKCS #1 v2.1 [69] as explained next. The standardized construction RSA-OAEP requires two random hash functions G and H with small input size (less than the RSA modulus) but arbitrarily sized outputs. Both these hash functions are instantiated in PKCS by the MGF1 pseudo-random number generator [69]. On input x, MGF1 uses a hash function h in counter mode: $\mathrm{MGF1}(x) = h(x||count0)||h(x||count1)||h(x||count2)||\ldots$, where h is either SHA-1 or SHA-2. Because MGF1 is not a regular standardized hash function, we use a term "non-standard hash function" for such functions. These functions instantiate a hash function of arbitrary output size by utilizing a standard fixed length hash function such as SHA-1 or SHA-2. Similarly, in other OAEP-type schemes, instantiation of such hash functions is done by using similar "non-standard hash functions".

OAEP-type schemes (OAEP, OAEP+,OAEP-3R) discussed above, work only

scheme with probability close to 1.

for restricted message length (less than input size of trapdoor one-way permutation) except OAEP-4X, which can process long messages (more than input size of trapdoor one-way permutation) as well. To encrypt lengthy messages, OAEP-4X uses one extra hash function and a semantically secure symmetric encryption scheme along with four hash functions. In OAEP-4X, the ability of handling long messages is the result of utilizing the well known Tag-KEM/DEM framework [1,25]. Tag-KEM/DEM is considered a hybrid encryption scheme [1,7,43,47,61,62,67,81]. In the hybrid paradigm, an asymmetric key encapsulation mechanism (KEM) combines with a symmetric data encapsulation mechanism (DEM). Traditionally, KEM is a probabilistic algorithm that produces a random symmetric key and an asymmetric encryption of that key as the key encapsulation. DEM is a deterministic algorithm that takes a symmetric key, generated by KEM, and encrypts the message under that key. In Tag-KEM/DEM framework, KEM takes a feedback, referred to as the 'tag', from DEM part and then generates key encapsulation. Final ciphertext results from concatenation of key encapsulation and encryption of message. This traditional hybrid paradigm suffers from high ciphertext overhead (difference between plaintext and ciphertext length) equal to the size of asymmetric encryption of key.

3.1.2 Motivation

All the previous OAEP-based encryption schemes require a perfect random function, i.e. a random oracle, over an arbitrary domain and/or arbitrary range. However, in practice one has access to a random function or permutation over a relatively small domain/range only, such as block-ciphers and hash functions. To solve the problem of generating lengthy hash outputs, RSA-Full Domain Hash [12, 14, 38] or the Mask Generation Function (MGF1) [69] in RSA-OAEP are currently implemented with a complex construction of fixed length hashes and counters. When a fixed length hash function is used with an input of m blocks and the requirement is to produce an n-block output, the hash function has to run approximately $m \times n$ times. All of the previously mentioned schemes proven secure in ROM (OAEP, OAEP+, OAEP-3R, OAEP-4X) require one or more hash functions with output size larger than standard sizes (e.g., SHA-1, SHA-512). While the security analysis of these schemes treat the hash functions as random oracles, the works [10, 70] showed that the hash function instantiation proposed in the literature for such cases are weaker than a random oracle. "Non-standard hash functions" (such as MGF1) are not well analyzed in literature, have complex construction of fixed length hash functions with counter and are also proven weaker than random oracle. This raises a question on the possibility of modifying the OAEP framework which does not require any "non-standard hash function" and where all the computations are performed in standardized input-output settings.

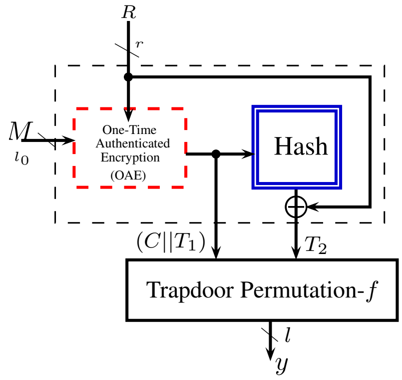
Development of schemes from OAEP to OAEP-4X shows differences in the number of rounds, depending upon calls to the hash functions used. OAEP and OAEP+ are considered as 2 round structures, OAEP-3R as 3 round and OAEP-4X as 4 round. This naturally poses a question on the possibility of further development of the OAEP-type scheme while reducing the number of rounds. As already remarked, OAEP-type constructions are good candidates for hybrid encryption to construct KEMs, as in [25]. Our motivation for this work also comes from an open problem mentioned in [1] about having a hybrid construction from different primitives like an ideal permutation.

Interestingly, popular Sponge constructions [20] based on iterative permutation is a suitable solution to all the problems mentioned earlier. In a Sponge function, for an m-block input and an n-block hash output, roughly $m + n$ calls to the internal primitive permutation are required. Moreover, the number of permutation calls in a Sponge function [20], used as a hash function, and SpongeWrap [21], a modification of Sponge function used as AE, are equal in general. Therefore, the versatility of Sponge function encourages the designers to come up with more useful and efficient design.

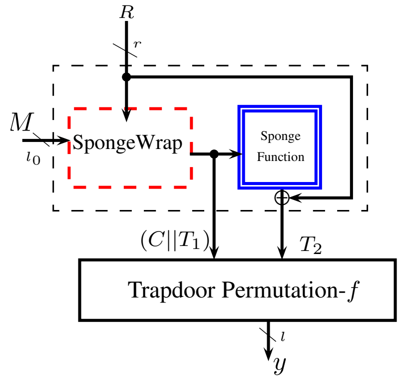
3.1.3 General View of OAEP+ with Sponge

In this section, we provide Sponge instantiated version of the *general view* of the OAEP+ discussed in section 1.2.3. This helps us to elaborate the basis of the design of our proposal SpAEP scheme. This *general view* is shown in Figure 3.1(a).

In this chapter, we provide f-SpAEP as an example of this general view where the f-SpAEP scheme uses SpongeWrap as *OAE* and a Sponge function as *Hash* part with different IV.



(a) Generic OAEP+ with f



(b) Sponge based view of Generic OAEP+

Figure 3.1: Generalization of OAEP+ to SpAEP

3.2 Contribution

In this work, we introduce a *Sponge based Asymmetric Encryption Padding* scheme (SpAEP), a novel way to use the SpongeWrap [21] and the Sponge function [20] to encrypt arbitrary length messages in the setting of public key cryptography. Both the functions SpongeWrap and Sponge use a public invertible permutation as a primitive function. Number of permutation calls in both Sponge function and SpongeWrap are generally the same for equal number of input-output data blocks.

- We provide a new approach to construct asymmetric key cryptographic schemes in ideal permutation model by utilizing permutations, having smaller/practical domain, in SpAEP. All the previous public key cryptography literature dealing with OAEP-based encryption are proven secure in Random Oracle Model that requires hash functions (or a random function) over an arbitrary domain.

- SpAEP uses the Sponge function and the SpongeWrap in standard input-output settings, proposed for "Sponge functions" [19–22], as per the security requirement. Therefore, SpAEP removes the requirement of having a "non-standard hash function", which is required in previous OAEP-type schemes(OAEP, OAEP+, OAEP-3R, OAEP-4X, etc.,).

- In SpAEP, both functions (Sponge function and SpongeWrap) are used in pipelined structure. After a fixed number of permutation calls of

SpongeWrap, both functions (Sponge function and SpongeWrap) are used in parallel fashion to speed-up the process. Therefore, we consider SpAEP as 1 round structure in comparison to other OAEP-type schemes. However, the functions are not parallelizable during decryption.

Features of SpAEP and comparison with other OAEP-type schemes

- Although the permutation used in Sponge is invertible, we do not use this fact for our construction and provide inverse-freeness during both encryption and decryption. Therefore our construction allows using permutations which are inefficient to invert but efficient in the forward direction. That is, computation time, implementation or memory efficiency of the forward direction of the permutation can be exploited by a user in our design. Moreover, our design allows using a non-invertible mapping in the Sponge function.

- Let f be a trapdoor one-way permutation then we denote the instantiation of our scheme with f by f-SpAEP. Our construction f-SpAEP can process arbitrary length messages and is CCA secure when used with any trapdoor one-way permutation.

- We provide a formal security proof of f-SpAEP in adaptively chosen ciphertext attack (CCA) setting in the ideal permutation model. Instead of directly using the random oracle model based security proof of Sponge construction, we provide a dedicated proof from scratch in ideal permutation model to avoid multi-stage game problem [5,78]. Usage of ideal permutation model allows stronger adversary in consideration of security proof, where the adversary have access to both underlying permutation and function based on this permutation. This stronger adversary in not applicable in RO model because function is itself considered as a black-box. Although [86] introduced an efficient scheme in ideal permutation model with full domain permutation encryption, it is still impractical due to the hardness of having large sized permutation (the needed size of the permutation is equal to the size of the trapdoor one-way permutation itself). A similar problem comes up when a scheme requires hash outputs which are different (generally larger) than the output size of standard hash functions.

	OAEP [13]	OAEP+ [98]	OAEP-3R [86]	OAEP-4X [2]	SpAEP
Ciphertext-overhead	$3t$	$3t$	$2t$	t	$2t$
# Function calls	2 Hash	3 Hash	3 Hash	5 Hash, 1 Symmetric Encryption (E)	1 SpongeWrap, 1 Sponge function
# Function calls Sequential or Parallel (Encryption)	Sequential	2 parallel, 1 sequential	Sequential	Mixed	Parallel
Trapdoor Perm.-f	RSA, Rabin	Any f	Any f	Any f	Any f
Max. Message size with f	$\ell - 3t$	$\ell - 3t$	$\ell - 2t$	Any	Any

Table 3.1: Comparison of OAEP, OAEP+, OAEP-3R, SpAEP, OAEP-4X. The security parameter in term of number of bits is denoted by t. That is, the number of queries required in order to break the scheme with probability 1 is 2^t. The input-output size of trapdoor one way permutation f is denoted by ℓ.

In Table 3.1, we compare OAEP [13], OAEP+ [98], OAEP-3R [86] and OAEP-4X [2] with SpAEP. The ciphertext overhead values in the table are taken from Table 1 in [2].

OAEP, OAEP+ and OAEP-3R can only handle messages of length less than input size of trapdoor one-way permutation while OAEP-4X and SpAEP can handle any message size.

Next we provide some comments on Table 3.1. The 2 hash function calls for OAEP are sequential and so are the 3 hash function calls for OAEP-3R. Out of the 3 hash function calls in OAEP+, only 2 can run in parallel. In OAEP-4X, for messages having size less than the input size of the trapdoor one-way permutation, 4 hash function calls are required sequentially. For long messages (message size more than input size of trapdoor one-way permutation), OAEP-4X uses 5 hash functions (H1, H2, H3, H4, and G) and one symmetric encryption scheme (E). Initially, only two hash function calls run in parallel (H1,G) then H2 and E runs parallel, and then H3 and H4 runs sequentially. Overall in OAEP-4X, for long messages, only two functions calls can run in parallel at any instant. From Table 3.1, we can clearly see that SpAEP is more efficient in comparison to other schemes. Although SpAEP has t-bits extra ciphertext overhead with respect to OAEP-4X, yet as explained earlier this is a minor concern in many applications.

One may argue that any improvement in OAEP-type scheme does not directly translate to an efficient public key encryption scheme because of the high computation time of the trapdoor one-way permutation, as against the OAEP structure. However, considering recent developments in lattice based cryptography [17, 77, 84], one can see that the computation time of trapdoor functions can be reduced significantly.

In summary, we are proposing an asymmetric padding scheme which is simpler and more efficient in terms of structure and functionality than the existing OAEP-type schemes.

3.3 SpAEP: Sponge based Asymmetric Encryption Padding

3.3.1 Description

SpAEP is a Sponge function based construction. SpAEP iterates a fixed permutation $\pi : \{0,1\}^r \times \{0,1\}^c \rightarrow \{0,1\}^r \times \{0,1\}^c$ similar to the Sponge construction and SpongeWrap [20–22].

The bit length of input and output of π, called bit rate, is $b = r + c$. The term r is called input rate and the term c is called capacity rate. The permutation π is the only underlying cryptographic primitive used by SpAEP. For using SpAEP for asymmetric key setting, one can use any family of trapdoor one-way permutations $\mathcal{F}$ such as RSA. The resulting scheme is called $\mathcal{F}$-SpAEP$^{\pi}$ or simply $\mathcal{F}$-SpAEP. The output of the encryption function f is $Y \in \{0,1\}^{\ell}$ and the inverse of f is represented by f^{-1}. The notation $\lfloor x \rfloor_k$ (resp. $\lceil x \rceil_k$) represents the first (resp. last) k bit of x. Figure 3.2 shows the graphical representation of SpAEP.

For security parameter k, a permutation π with parameters r and c can be chosen as explained in [20–22]. For the scheme to be simple and compatible with given input-output length ℓ of a trapdoor one-way permutation f, we assume that $\ell = n * r + 2k$ should hold for a positive integer $n \geq 1$. SpAEP uses a reversible padding function $pad(\cdot)$ to generate blocks of length r bits such that $|pad(x)| \geq \ell - 2r$ and $|pad(x)| \bmod r = 0$. For SpAEP, we use 10*1 reversible padding and define padding and unpadding function accordingly.

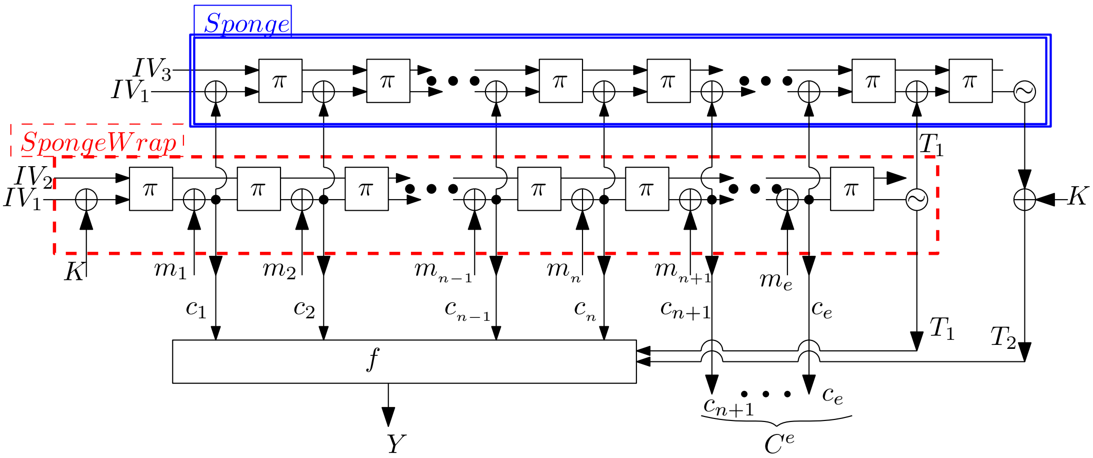


Figure 3.2: SpAEP with any trapdoor one way permutation f and public invertible permutation $\pi : \{0,1\}^r \times \{0,1\}^c \leftarrow \{0,1\}^r \times \{0,1\}^c$. SpAEP accepts message M and internally calls pad(M)=$m_1||\ldots||m_n||m_{n+1}||\ldots||m_e$ such that $n = (\ell - 2k)/r$, $|pad(M)| \geq (\ell - 2k)$ and $|m_1| = |m_2| = \ldots = r$. The size of the trapdoor permutation-f is denoted by ℓ and the size of random K and Tags T_1 and T_2 is k-bit. The symbol $\circlearrowleft$ represents taking k-bit output from r-bit input.

<u>$pad(x)$</u>

$$\begin{cases} x||10^{((\ell-2k)-|x|-2)}1, & \text{if} |x| \leq (\ell - 2k - 2) \\ x||10^{(r-1)}1, & \text{if } |x| = \ell - 2k - 1 \\ x||1||0^{r-((|x|+1) \bmod r)-1}||1, & \text{otherwise} \end{cases}$$

<u>$unpad(Y)$</u>

if $\exists\ x \neq \emptyset\ s.t.\ Y = x||1||0^z||1\ where$
$0 \leq z \leq \ell - 2k - 3\ if\ |Y| = \ell - 2k\ OR$
$0 \leq z \leq r - 1\ if\ |Y| > \ell - 2k$ **then**
  return x
else
  return $\perp$

On input M, SpAEP internally computes $pad(M) = m_1||\ldots||m_n||m_{n+1}||\ldots||m_e$ and produces output $c_1||\ldots||c_n||c_{n+1}||\ldots||c_e$ along with k-bit tags T_1 and T_2, where $n \geq 1, e > n$. We denote $C^f = c_1||\ldots||c_n||T_1||T_2$ and $C^e = c_{n+1}||\ldots||c_e$. Final output of the $\mathcal{F}$-SpAEP will be $Y||C_e$, where $Y = f((C^f)$, and $C_e = c_{n+1}||\ldots||c_e$.

Encryption and Decryption in $\mathcal{F}$-SpAEP are described in Algorithms 1 and 2 respectively. Note that the encryption and decryption procedures of SpAEP use only the forward direction of the permutation. Therefore, we can have a permutation that is more efficient in forward direction compared to its inverse.

Algorithm 1: Encryption: $SpAEP - E_f^{\pi}(M) = Y||C^e$

Initialization:
$IV_1 = 0^r$,$IV_2 = 0^c$,$IV_3 = IV_2 \oplus 1$,
$w = IV_2, x = IV_1$
Random Nonce: $K \xleftarrow{\$} \{0,1\}^k$
$pad(M) = m_1||m_2||\ldots||m_e$, where $|m_i| = r \ \forall 1 \leq i \leq e$
$x = x \oplus K||0^{r-k}$
for $i = 1 \rightarrow e$ **do**
    $(x||w) = \pi(x||w)$
    $x = x \oplus m_i$
    $c_i = x$
$(x||w) = \pi(x||w)$; $T_1 = \lfloor x \rfloor_k$
$x = IV_1$ and $w = IV_3$
for $i = 1 \rightarrow e$ **do**
    $x = x \oplus c_i$
    $(x||w) = \pi(x||w)$
$x = x \oplus T_1||0^{r-k}$
$(x||w) = \pi(x||w)$
$T_2 = \lfloor x \rfloor_k \oplus K$
$C^f = c_1||c_2||\ldots||c_n||T_1||T_2$;
$C^e = c_{n+1}||\ldots||c_e$
$Y = f(C^f)$
Return: $Y||C^e$

Algorithm 2: Decryption: $SpAEP - D_{f^{-1}}^{\pi}(Y||C^e) = M \ or \ \perp$

Initialization:
$IV_1 = 0^r$,$IV_2 = 0^c$,$IV_3 = IV_2 \oplus 1$
$C^f = c_1||c_2||\ldots||c_n||T_1||T_2 = f^{-1}(Y)$;
$c_{n+1}||\ldots||c_e = C^e$, $w = IV_3, x = IV_1$
for $i = 1 \rightarrow e$ **do**
    $x = x \oplus c_i$
    $(x||w) = \pi(x||w)$
$x = x \oplus T_1||0^{r-k}$
$(x||w) = \pi(x||w)$; $K = \lfloor x \rfloor_k \oplus T_2$
$x = K||0^{r-k}$; $w = IV_2$
for $i = 1 \rightarrow e$ **do**
    $(x||w) = \pi(x||w)$
    $m_i = x \oplus c_i$
    $x = c_i$
$(x||w) = \pi(x||w)$; $T_1' = \lfloor x \rfloor_k$
if $T_1 = T_1'$ **then**
    if $\exists\ M \ s.t.$ $M = unpad(m_1||\ldots||m_e)$ **then**
        Return: M
    else
        Return: $\perp$
else
    $\perp$.

3.3.2 CCA Security of $\mathcal{F}$-SpAEP

Next we provide a formal proof of CCA security of $\mathcal{F}$-SpAEP. As described in Section 2.2, the experiment that the adversary $\mathcal{A}$ runs against the scheme $\mathcal{F}$-SpAEP is the following.

Experiment: $Exp^{ind-cca-d}_{\mathcal{F}-SpAEP^{\pi},\mathcal{A}}(k)$

---

1. $(\underbrace{f}_{\textsf{pk}}, \underbrace{f^{-1}}_{\textsf{sk}}) \xleftarrow{\$} \textsf{GenEnc}(1^r)$;
2. $(M_0, M_1, s) \leftarrow \mathcal{A}^{\pi(\cdot), SpAEP-D^{\pi}_{f^{-1}}(\cdot), f(\cdot)}$;
3. $Y^*||C^{e*} \leftarrow SpAEP - E^{\pi}_f(M_d)$;
4. $d' \leftarrow \mathcal{A}^{\pi(\cdot), SpAEP-D^{\pi}_{f^{-1}}(\cdot), f(\cdot)}(M_0, M_1, Y^*||C^{e*}, s)$;
5. return d';

where $SpAEP-D^{\pi}_{f^{-1}}(\cdot)$ is decryption oracle and $SpAEP-E^{\pi}_f(\cdot)$ is encryption oracle.

Theorem 1. *If the underlying trapdoor permutation f, generated using trapdoor generator $\mathcal{F}$, is one way, then $\mathcal{F}$-SpAEP is secure against adaptive chosen ciphertext attack in the Ideal permutation model. The success probability of adversary $\mathcal{A}$ for CCA attack is*

$$Pr[Exp^{ind-cca-d}_{\mathcal{F}-SpAEP,\mathcal{A}}(k) = d] \leq \frac{1}{2} + \frac{(q-1)q}{2^{b+1}} + \frac{q(q+1)}{2^c} + \frac{5q_D}{2^k} + \frac{q_{\pi_{\mathcal{A}}}}{2^k} + Adv^{ow}_{\mathcal{F}}(\mathcal{B}_{\mathcal{A}}\ Succeeds) + \frac{q_{\pi_{\mathcal{A}}}}{min(2^k, 2^c)},$$

where $q = q_\pi + q_{\pi^{-1}}$, q_π and $q_{\pi^{-1}}$ are the number of π and π^{-1} queries respectively, $q_{\pi_{\mathcal{A}}}$ is the number of π and π^{-1} queries by $\mathcal{A}$, q_D is the number of decryption queries and (b, c, k) are parameters of permutation π as defined earlier, $\mathcal{B}$ is an adversary that finds the complete random input C^f of trapdoor-one way permutation f given $Y \xleftarrow{\$} \{0,1\}^{\ell}$ such that $Y = f(C^f)$, without having knowledge of f^{-1}. Adversary $\mathcal{B}$ uses $\mathcal{A}$ as a subroutine internally. $Adv^{ow}_{\mathcal{F}}(\mathcal{B}_{\mathcal{A}}\ Succeeds)$ is the success advantage that a particular adversary $\mathcal{B}$ has in breaking the trapdoor one-way permutation f of $\mathcal{F}$. The time and space requirements of $\mathcal{B}$ are related to $\mathcal{A}$ as follows:

$$Time(\mathcal{B}) = O(Time(\mathcal{A}) + q_{\pi_{\mathcal{A}}} \cdot t_f + (q_{\pi_{\mathcal{A}}} + q_D) \cdot t_f);$$

$$Space(\mathcal{B}) = O(Space(\mathcal{A}) + q_{\pi_{\mathcal{A}}} \cdot \ell). \tag{3.1}$$

Here, t_f is the time required to compute f, and space is measured in the number of storage bits.

Proof. We will use Game based playing technique [15, 16]. We start from the original CCA game as discussed in Section 3.3.2. Let $Exp_{\mathcal{F}-SpAEP,\mathcal{A}}$ Or $Exp^{ind-cca-d}_{\mathcal{F}-SpAEP,\mathcal{A}}(k) = d$ denote the event that $\mathcal{A}$ outputs $d' = d$ where $d \xleftarrow{\$} \{0,1\}$. We want to show that $\mid Pr[Exp_{\mathcal{F}-SpAEP,\mathcal{A}}] \mid = \frac{1}{2} + negl(k)$. We slightly change $\mathcal{F}$-SpAEP into a sequence G0, G1, ..., G12 such that:

$$\boxed{\begin{array}{l} Pr[Exp_{\mathcal{F}-SpAEP,\mathcal{A}}] = Pr[Exp_{G0,\mathcal{A}}] \\ Pr[Exp_{G(i-1),\mathcal{A}}] = Pr[Exp_{Gi,\mathcal{A}}] + negl(\ell)\ \forall 1 \leq i \leq 11 \\ Pr[Exp_{G12,\mathcal{A}}] = \frac{1}{2} \end{array}}$$

Each game has the following functions:

- *Encryption* (*Enc*), *Decryption*(*Dec*): perform Encryption and Decryption,
- π, π^{-1}: public invertible permutation and its inverse,
- π_{Enc}: permutation π calls by encryption and decryption functions,
- $\pi_{\mathcal{A}}$, $\pi^{-1}_{\mathcal{A}}$: permutation π, π^{-1} calls by adversary $\mathcal{A}$.

Encryption, *Decryption*, $\pi_{\mathcal{A}}$ and $\pi^{-1}_{\mathcal{A}}$ are public oracles, which are also accessible to the adversary. In each game, the following sets are maintained: I_π by π and π^{-1}, I_{Enc} by π_{Enc} and $I^{\mathcal{A}}_{\pi}$ by $\pi_{\mathcal{A}}$ and $\pi^{-1}_{\mathcal{A}}$ to store input-output relations.

Another set $\mathcal{L}_c : \{g : g \in \{0,1\}^c\}$ is also maintained internally by π and π^{-1} for storing capacity bits. The set $\mathcal{L}_c$ is initialized to $\{IV_2, IV_3\}$ because IV_2 is the capacity part of the input to first π of OAE (SpongeWrap) part and IV_3 is the capacity part of the input to the first π of *Hash* (Sponge) part. The set $\mathcal{L}_c$ is updated on every call to π. Precisely, two c-bit values are appended to $\mathcal{L}_c$ on each π call. These two values are the capacity bits of the inputs and output of π.

Note that $q = q_\pi + q_{\pi^{-1}}$, $q_\pi = q_{\pi_{\mathcal{A}}} + q_{\pi_{Enc}}$ and q_D= number of decryption queries. Further, the encryption query has $2(e+1)$ calls to π_{Enc}.

In each of the games G0, G1, G2, G3, G4, G5 we make small incremental changes in the permutation to have response in some particular fashion. In games G6, G7, we make changes in the *Decryption* oracle and make it independent of f.

Finally, in games G8, G9, G10, G11, G12 we make changes in *Encryption* oracle along with some changes in $\pi_{\mathcal{A}}$ oracle to achieve that d of M_d is independent of all previous queries. We represent the *Hash* part of SpAEP as a function $H^{\pi}(j_1, j_2, j_3..., j_i, j_{i+1})$ whose output J is such that

$$J||* = \pi_{Enc}\Big(\pi_{Enc}(\dots(\pi_{Enc}(\pi_{Enc}(j_2||0^c \oplus j_1) \oplus j_3||0^c) \oplus j_4||0^c) \dots \oplus j_{i-1}||0^c) \oplus j_i||0^{b-k}\Big) \oplus j_{i+1}||0^{b-k}$$

where π is b-bit permutation, $j_1 \in \{0,1\}^b$, $(j_2, j_3, \dots j_{i-1}) \in \{0,1\}^r$, $(J, j_i, j_{i+1}) \in \{0,1\}^k$ and $* \in \{0,1\}^c$.

Game G0: This game perfectly simulates the $\mathcal{F}$-SpAEP.

$$\Pr[Exp_{\mathcal{F}-SpAEP,\mathcal{A}}]=\Pr[Exp_{G0,\mathcal{A}}].$$

Game G0: Initialize $I_\pi = I_{Enc} = I_\pi^{\mathcal{A}} = \emptyset$, $IV_1 = 0^r$ $IV_2 = 0^c, IV_3 = IV_2 \oplus 1$

On Encryption-Query M_d

1. $x = IV_1$ and $w = IV_2$
2. Random Nonce: $K \xleftarrow{\$} \{0,1\}^k$
3. $pad(M) = m_1||m_2||\ldots||m_e$, where $|m_i| = r \ \forall 1 \le i \le e$
4. $x = x \oplus K||0^{r-k}$
5. **for** $i = 1 \to e$ **do**
   - $(x||w) = \pi(x||w)$
   - $x = x \oplus m_i$
   - $c_i = x$
6. $(x||w) = \pi(x||w)$; $T_1 = \lfloor x \rfloor_r$
7. $x = IV_1$ and $w = IV_3$
8. **for** $i = 1 \to e$ **do**
   - $x = x \oplus c_i$
   - $(x||w) = \pi(x||w)$
9. $x = x \oplus T_1||0^{r-k}$
10. $(x||w) = \pi(x||w)$
11. $T_2 = \lfloor x \rfloor_k \oplus K$
12. $C^f = c_1||c_2||\ldots||c_n||T_1||T_2$
13. $C^e = c_{n+1}||\ldots||c_e$
14. $Y = f(C^f)$
15. Return: $y||C^e$

On Decryption-Query $Y||C^e$

1. $x = IV_1$ and $w = IV_2$
2. $C^f = c_1||c_2||\ldots||c_n||T_1||T_2 = f^{-1}(y)$; $c_{n+1}||\ldots||c_e = C^e$
3. $C = c_1||c_2||\ldots||c_n||T_1||T_2||c_{n+1}||\ldots||c_e$, where $|c_i| = r$ for $1 \le i \le e$, $|T_1| = |T_2| = k$
4. **for** $i = 1 \to e$ **do**
   - $x = x \oplus c_i$
   - $(x||w) = \pi(x||w)$
5. $x = x \oplus T_1||0^{r-k}$
6. $(x||w) = \pi(x||w)$; $K = \lfloor x \rfloor_k \oplus T_2$
7. $x = K||0^{r-k}$; $w = IV_2$
8. **for** $i = 1 \to e$ **do**
   - $(x||w) = \pi(x||w)$
   - $m_i = x \oplus c_i$
   - $x = c_i$
9. $(x||w) = \pi(x||w)$; $T_1' = \lfloor x \rfloor_k$
10. **if** $T_1 = T_1'$ **then**
    - **if** $\exists$ M *s.t.* $M = unpad(m_1||\ldots||m_e)$ **then**
      - Return:M
    - **else**
      - Return: $\perp$

    else
    - $\perp$.

On π-Query m, where $m \in \{0,1\}^b$

1. let $(x||w) = m$, where $x \in \{0,1\}^r$, $w \in \{0,1\}^c$,
2. if $(m, v) \in I_\pi$ then return v
3. $v \xleftarrow{\$} \{0,1\}^b$
4. if $\exists$ m' s.t $(m', v) \in I_\pi$, then $v \xleftarrow{\$} \{0,1\}^b \setminus \{v : (*, v) \in I_\pi\}$, where $* \in \{0,1\}^b$
5. $I_\pi = I_\pi \bigcup \{(m, v)\}$
6. return v;

On π^{-1}-Query $v = \{v_1||v_2\}$. where $v_1 \in \{0,1\}^r,\ v_2 \in \{0,1\}^c,\ v \in \{0,1\}^b$

1. if $(m, v) \in I_\pi$ then return m
2. $m \xleftarrow{\$} \{0,1\}^b$
3. if $\exists$ v' s.t $(m, v') \in I_\pi$, then $m \xleftarrow{\$} \{0,1\}^b \setminus \{m : (m, *) \in I_\pi\}$, where $* \in \{0,1\}^b$
4. $I_\pi = I_\pi \bigcup \{(m, v)\}$
5. return m;

On π_{Enc}-Query m

1. $v = \pi(m)$
2. $I_{Enc} = I_{Enc} \bigcup \{(m, v)\}$
3. return v;

On $\pi_{\mathcal{A}}$-Query m

1. $v = \pi(m)$
2. $I_\pi^{\mathcal{A}} = I_\pi^{\mathcal{A}} \bigcup \{(m, v)\}$
3. return v;

On $\pi_{\mathcal{A}}^{-1}$-Query v

1. $m = \pi^{-1}(v)$
2. $I_\pi^{\mathcal{A}} = I_\pi^{\mathcal{A}} \bigcup \{(m, v)\}$
3. return v;

Figure 3.3: Game G0 of $\mathcal{F}$-SpAEP

Game G0 and Game G1: The response of both these games is exactly same. In the permutation, both games choose their response randomly while excluding the previous responses, in order to satisfy the permutation property. In G1, there is an addition of dummy lines, shown in dash boxes, in which if random chosen response v is already with some m' as $\{m', v\} \in I_\pi$ then we say $bad \leftarrow true$.

$$\Pr[Exp_{G0,\mathcal{A}}]=\Pr[Exp_{G1,\mathcal{A}}].$$

Game $\boxed{\textbf{G1}}$ **G2:** Initialize $I_{enc} = I_\pi = I_\pi^{\mathcal{A}} = \emptyset$, $IV_1 = 0^r, IV_2 = 0^c$, $IV_3 = IV_2 \oplus 1$.

On π**-Query** m,where $m \in \{0,1\}^b$	**On** π^{-1}**-Query** v, where $v \in \{0,1\}^b$
1. let $(x\|\|w)$=m,where $x \in \{0,1\}^r$, $w \in \{0,1\}^c$,	1. let $(v_1\|\|v_2)$=v,where $v_1 \in \{0,1\}^r$, $v_2 \in \{0,1\}^c$,
2. if $(m,v) \in I_\pi$ then return v	2. if $(m,v) \in I_\pi$ then return m
3. $v \xleftarrow{\$} \{0,1\}^b$	3. $m \xleftarrow{\$} \{0,1\}^b$
4. if $\exists$ m' s.t $(m',v) \in I_\pi$, then **bad**$_{\mathbf{b}} \leftarrow$ **true** and $\boxed{v \xleftarrow{\$} \{0,1\}^b \setminus \{v : (*,v) \in I_\pi\}}$, where $* \in \{0,1\}^b$	4. if $\exists$ v' s.t $(m,v') \in I_\pi$, then **bad**$_{\mathbf{b}} \leftarrow$ **true** and $\boxed{m \xleftarrow{\$} \{0,1\}^b \setminus \{m : (m,*) \in I_\pi\}}$, where $* \in \{0,1\}^b$
5. $I_\pi = I_\pi \bigcup \{(m,v)\}$	5. $I_\pi = I_\pi \bigcup \{(m,v)\}$
6. return v;	6. return m;

Rest of Oracles same as G0

Figure 3.4: Game G1 and G2: G1 has dummy line, shown in dash box, along with solid line box same as compare to G0. G2 includes dash box line but without solid line box.

Game G1 and Game G2: In Game G2, π avoid checking of previous response. Both G1 and G2 behave the same till $\mathbf{bad_b}$ occurs. The event $\mathbf{bad_b}$ occurs when a collision over b-bit outputs of permutation π takes place and is corrected in G1 but not in G2. If q is the total number of queries to π and π^{-1}, then $\Pr[\mathbf{bad_b}]$ is $\leq \frac{q(q-1)}{2^{b+1}}$.

$$|Pr[Exp_{G2,\mathcal{A}}] - Pr[Exp_{G1,\mathcal{A}}]| = \Pr[\mathbf{bad_b}] \leq \frac{(q-1)q}{2^{b+1}}.$$

Game G2 and Game G3: Output of both game G2 and G3 are the same where the output of π is not checked for collision over previous responses. In addition, G3 maintains a new list $\mathcal{L}_c$, initialized with IV_2 and IV_3, which stores the capacity part of input-output of π. G3 also adds a dummy line of code, shown in dash box, in which if output of π has collision over capacity part of previous responses in list $\mathcal{L}_c$, then $\mathbf{bad_c}$ happens as **true**.

$$|Pr[Exp_{G3,\mathcal{A}}] = Pr[Exp_{G2,\mathcal{A}}]|.$$

Game G3 $\boxed{\textbf{G4}}$: Initialize $\mathcal{L}_c = I_{enc} = I_\pi = I_\pi^{\mathcal{A}} = \emptyset$, $IV_1 = 0^r, IV_2 = 0^c$, $IV_3 = IV_2 \oplus 1$.

On π-Query m, where $m \in \{0,1\}^b$	**On π^{-1}-Query** v. where $v \in \{0,1\}^b$
1. let $(x\|\|w)$=m,where $x \in \{0,1\}^r$, $w \in \{0,1\}^c$,	1. let $(v_1\|\|v_2)$=v,where $v_1 \in \{0,1\}^r$, $v_2 \in \{0,1\}^c$,
2. if $(m, v) \in I_\pi$ then return v	2. if $(m, v) \in I_\pi$ then return m
3. $v_1\|\|v_2 \xleftarrow{\$} \{0,1\}^b$,where $v_1 \in \{0,1\}^r$, $v_2 \in \{0,1\}^c$,	3. $m'\|\|m'' \xleftarrow{\$} \{0,1\}^b$,where $m' \in \{0,1\}^r$, $m'' \in \{0,1\}^c$,
4. if $v_2 \in \mathcal{L}_c \bigcup \{w\}$, then **bad**$\leftarrow$**true** and $\boxed{v_2 \xleftarrow{\$} \{0,1\}^c \setminus \mathcal{L}_c \bigcup \{w\}}$	4. if $m'' \in \mathcal{L}_c \bigcup \{v_2\}$, then **bad**$\leftarrow$**true** and $\boxed{m'' \xleftarrow{\$} \{0,1\}^c \setminus \mathcal{L}_c \bigcup \{v_2\}}$
5. $I_\pi = I_\pi \bigcup \{(m, v_1\|\|v_2)\}$ and $\mathcal{L}_c = \mathcal{L}_c \bigcup \{v_2, w\}$	5. $I_\pi = I_\pi \bigcup \{(m'\|\|m'', v)\}$ and $\mathcal{L}_c = \mathcal{L}_c \bigcup \{m'', v_2\}$
6. return $v = v_1\|\|v_2$;	6. return $m = m'\|\|m''$;

Rest of Oracles same as G0

Figure 3.5: Game G3 and G4: G3 includes line of code in dash-box, but not of solid-box. G4 includes dash-box and solid-box.

Game G3 and Game G4: In G4, if $\mathbf{bad_c}$ happens then capacity part of output is chosen again randomly but avoiding the previous responses in $\mathcal{L}_c$. Both the games are the same till $\mathbf{bad_c}$ occurs. The event $\mathbf{bad_c}$ occurs if there is a collision over c-bit output of permutation π. Pr[$\mathbf{bad_c}$] is $\leq \frac{q(q+1)}{2^c}$.

$$|Pr[Exp_{G4,\mathcal{A}}] - Pr[Exp_{G3,\mathcal{A}}]| = \Pr[\mathbf{bad_c}] \leq \frac{q(q+1)}{2^c}.$$

Game G4 and Game G5: Output of π in G4 and G5 is same. In G5, code of π is re-written without any bad event.

$$|Pr[Exp_{G5,\mathcal{A}}] = Pr[Exp_{G4,\mathcal{A}}]|.$$

Game G5: Initialize $I_{enc} = I_\pi = I_\pi^{\mathcal{A}} = \emptyset$, $IV_1 = 0^r, IV_2 = 0^c$, $IV_3 = IV_2 \oplus 1$, $\mathcal{L}_c = \{IV_2, IV_3\}$.

On π-Query m, where $m \in \{0,1\}^b$

1. let $(x||w)$=m,where $x \in \{0,1\}^r$, $w \in \{0,1\}^c$,
2. if $(m, v)$$\in I_\pi$ then return v
3. $v_1||v_2 \xleftarrow{\$} \{0,1\}^b$,where $v_1 \in \{0,1\}^r$, $v_2 \in \{0,1\}^c$,
4. if $v_2 \in \mathcal{L}_c \bigcup \{w\}$, then $v_2 \xleftarrow{\$} \{0,1\}^c \setminus \mathcal{L}_c \bigcup \{w\}$
5. $I_\pi = I_\pi \bigcup \{(m, v_1||v_2)\}$ and $\mathcal{L}_c = \mathcal{L}_c \bigcup \{v_2, w\}$
6. return $v = v_1||v_2$;

On π^{-1}-Query v. where $v \in \{0,1\}^b$

1. let $(v_1||v_2)$=v,where $v_1 \in \{0,1\}^r$, $v_2 \in \{0,1\}^c$,
2. if $(m, v)$$\in I_\pi$ then return m
3. $m'||m'' \xleftarrow{\$} \{0,1\}^b$,where $m' \in \{0,1\}^r$, $m'' \in \{0,1\}^c$
4. if $m'' \in \mathcal{L}_c \bigcup \{v_2\}$, then $m'' \xleftarrow{\$} \{0,1\}^c \setminus \mathcal{L}_c \bigcup \{v_2\}$
5. $I_\pi = I_\pi \bigcup \{(m'||m'', v)\}$ and $\mathcal{L}_c = \mathcal{L}_c \bigcup \{m'', v_2\}$
6. return $m = m'||m''$;

Rest of Oracles same as G0

Figure 3.6: Game G5: same as G4 with simplified straight code

Game G5 and Game G6: Both the games are same. In Game G6 only a dummy operation, shown as dash-box, of $flag \leftarrow new$ is added in the *Decryption* oracle to denote a new query. The query is new in the sense that neither the query nor any part of the query during internal calls to π, of *Decryption* oracle, was queried earlier by the adversary. That is, query $\notin I_\pi^{\mathcal{A}}$. In decryption oracle there is addition of one more dummy line of bad_π as true if $T = T'$ happens for $flag = new$.

$$|Pr[Exp_{G6,\mathcal{A}}] = Pr[Exp_{G5,\mathcal{A}}]|.$$

Game G6 and **G7**: Initialize $I_\pi = I_{Enc} = I_{Dec} = I_\pi^{\mathcal{A}} = \emptyset$, $IV_1 = 0^r$ $IV_2 = 0^c, IV_3 = IV_2 \oplus 1,\ \mathcal{L}_c = \{IV_2, IV_3\}$; $flag \in \{old, new\}$

On Decryption-Query $Y||C^e$

1. $C^f = c_1||c_2||\ldots||c_n||T_1||T_2 = f^{-1}(y)$; $c_{n+1}||\ldots||c_e = C^e$
2. $x = IV_1$; $w = IV_3$; $flag \leftarrow old$
3. **for** $i = 1 \rightarrow e$ **do**
    - $x = x \oplus c_i$
    - If $\nexists\ v$ s.t. $(x||w, v) \in I_\pi^{\mathcal{A}}$ then $flag \leftarrow new$
    - $(x||w) = \pi_{Dec}(x||w)$
4. If $\nexists\ v$ s.t. $(x \oplus T_1||w, v) \in I_\pi^{\mathcal{A}}$ then $flag \leftarrow new$
5. $(x||w) = \pi_{Dec}((x \oplus T_1||0^{b-r})||w)$
6. $K = \lfloor x \rfloor_r \oplus T_2$; $x = K||0^{b-r}$ $w = IV_2$
7. **for** $i = 1 \rightarrow e$ **do**
    - If $\nexists\ v$ s.t. $(x||w, v) \in I_\pi^{\mathcal{A}}$ then $flag \leftarrow new$
    - $(x||w) = \pi_{Dec}(x||w)$
    - $m_i = x \oplus c_i$
    - $x = c_i$
8. If $\nexists\ v$ s.t. $(x||w, v) \in I_\pi^{\mathcal{A}}$ then $flag \leftarrow new$
9. $(x||w) = \pi_{Dec}(x||w)$,$T_1' = \lfloor x \rfloor_r$
10. **if** $T_1 = T_1'$ $and\ flag \leftarrow new$ **then**
    - $bad_\pi \leftarrow true$
    - Return:M Return: $\perp$
11. **if** $T_1 = T_1'$ $and\ flag \leftarrow old$ **then**
    - **if** $\exists\ M\ s.t.\ M = unpad(m_1||\ldots||m_e)$ **then**
        - Return:M
    - **else**
        - Return: $\perp$

    else
    - $\perp$.

Rest of oracles same as G5

Figure 3.7: Game G6 having extra lines of code as dummy, shown in dash box, compared to G5. G6 includes oval box not solid line box where Game G7 does not includes oval box but include solid line box

Game G8: Initialize $I_{enc} = I_\pi = I_\pi^{\mathcal{A}} = \emptyset$, $IV_1 = 0^r, IV_2 = 0^c$, $IV_3 = IV_2 \oplus 1$.
$\mathcal{L}_c = \{IV_2, IV_3\}$

On Decryption-Query $Y||C^e$

1 $w_0 = IV_2$;
2 If $\exists$ pad(M)=$m_1||m_2||\dots||m_e$, K, after $x_0 = K||0^{r-k} \oplus IV_1$ such that
$(x_0||w_0, v_{1_1}||v_{2_1}) \in I_\pi^{\mathcal{A}}$,
$(x_n||w_n, v_{1_{e+1}}||v_{2_{e+1}}) \in I_\pi^{\mathcal{A}}$,
$((z_{1_{e+1}} \oplus G)||u_{2_{e+1}}, z_{1_{e+2}}||z_{2_{e+2}}) \in I_\pi^{\mathcal{A}}$,
$f(c_1||\dots||c_e||\lfloor v_{1_{e+1}} \rfloor_k||T_2) = Y$, where $G = \lfloor v_{1_{e+1}} \rfloor_r||0^{r-k}$,
$T_2 = \lfloor z_{1_{e+2}} \rfloor_k \oplus K$
then **return** M
else **Return** $\perp$

Rest of Oracles same as G7

Following special notations is used during Game G8 and onwards in decryption oracle:

1. During *OAE* part of SpAEP, we represent input-output relation of π's subsequent calls for $pad(M) = m_1||\dots||m_e$ by $(v_{1_{i+1}}||v_{2_{i+1}}) = \pi(x_i||w_i)$, where $x_i = v_{1_i} \oplus \{m_i\}$, $w_i = v_{2_i}$ $0 \le i \le e$, $v_{1_0} = IV_1$,$m_0 = K$, $w_0 = IV_2$, $v_{1_i}, x_i \in \{0,1\}^r$ and $v_{2_i}, w_i \in \{0,1\}^c$. Then c_i will represent $m_i \oplus v_{1_i}$, where $1 \le i \le e$.

2. Input-output relation of π's subsequent call during *Hash* part of SpAEP will be represented as follows: $(z_{1_i+1}||z_{2_i+1}) = \pi(u_{1_i}||u_{2_i})$, $u_{1_i} = c_i \oplus z_{1_i}$, $u_{2_i} = z_{2_i}$, where $1 \le i \le (e+2)$, $u_{2_1} = IV_3$, $z_{1_1} = IV_1$, $c_{e+1} = T_1$, $c_{e+2} = K$

Figure 3.8: Game G8: Output of decryption oracle in G8 is same as G7 in re-written form and independent from sk or f^{-1}

Game G6 and Game G7: Both the games act similarly till bad_π occurs. The event bad_π occurs in *Decryption* oracle when a new query results in $T_1 = T_1'$

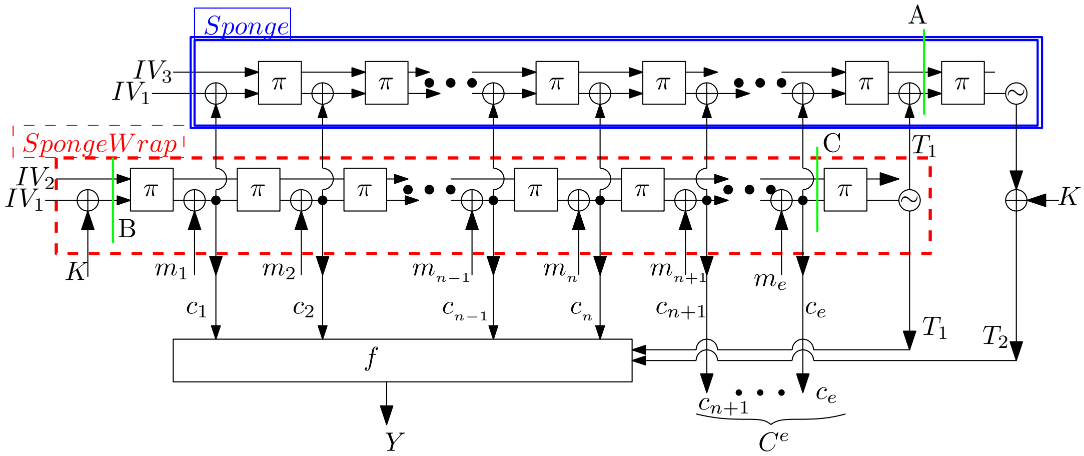


Figure 3.9: f-SpAEP with three checkpoints for testing if a *new* query has been made.

(mentioned in Algorithm 2 and Fig. 3.7). The bad_π event occurs with probability $\frac{5q_D}{2^k} + \frac{q_{\pi_\mathcal{A}}+q_{\pi^{-1}}}{2^k}$, as demonstrated below.

$$|Pr[Exp_{G7,\mathcal{A}}] - Pr[Exp_{G6,\mathcal{A}}]| = Pr[bad_\pi] \leq \frac{5q_D}{2^k} + \frac{q_{\pi_\mathcal{A}} + q_{\pi^{-1}}}{2^k}.$$

Let $(v_1||v_2) = \pi_{Dec}(x||w)$, where x,$v_1 \in \{0,1\}^r$ and w,$v_2 \in \{0,1\}^c$. In decryption, an input is a *new query* to π when $((x||w),(v_1||v_2)) \notin I_\pi^\mathcal{A}$ and *old query* when $((x||w),(v_1||v_2)) \in I_\pi^\mathcal{A}$. If a *new query* $(x||w)$ is input to π during decryption, then π outputs $v_1||v_2$, where $v_2 \notin \mathcal{L}_c$. That is, v_2 is also new. Since v_2 is unseen so far, it ensures that the input to the next call of π is certainly new. Further, since v_2 is new, next input $x'||v_2$ satisfies the condition $(x'||v_2,*) \notin I_\pi^\mathcal{A}$, where $*$ stands for any b bit value. Therefore one *new query* makes all subsequent inputs to $\pi(\cdot)$ as new. Any *new query* to π implies that a ciphertext Y queried to *Decryption* oracle has never been generated by the adversary. In Game G7, *Decryption* oracle return $\perp$ whenever adversary makes such a query.

To know if a *new query* has been made in SpAEP *Decryption* oracle, we consider three checkpoints, called A, B and C in Figure 3.9. Next we explain the situation when a bad_π event can occur in Game G7.

In *Hash*-part (*Sponge*), if any input before A is new, then A is also new as explained earlier. Hence a decryption query is certainly new if A is new. In the case of checkpoints B and C, it is not possible that B is *new query* and C is *old query*. This follows from our discussion above. Therefore we only need to check C to determine if there is a *new query* in the OAE part.

During encryption, let us denote the values at checkpoints A, B and C by $\alpha, K^*||0^{b-k}||IV_2$ and β respectively. Let $Y^*||C^{e*}$ be the target ciphertext and $C^* = C^{f*}||C^{e*}$ where $C^{f*} = c_1^*||\ldots||c_n^*||T_1^*||T_2^*$ and $C^{e*} = c_{n+1}^*||\ldots||c_e^*$ such that $Y^* = f(C^{f*})$.
The following cases cover all the possible cases for *new query*.

1. (A *new*, B *new*, C *new*): The bad_π event occurs only when tag $T_1 = T_1'$ (shown in Fig. 3.7)

   - : C$\neq \beta$: Then $T_1 = T_1'$ implies collision of the outputs of π over k-bit value. Probability of this event is $\frac{q_D}{2^k}$ for q_D queries to *Decryption* oracle
   - : C=β: Then $T_1 = T_1^*$ which means $c_i = c_i^*$ for all i such that $1 \leq i \leq n$ and $K = K^*$. This in turn results in $T_2 = T_2^*$. This leads to $C = C^*$, which is not allowed because adversary can not query $Y^* = f(C^*)$ to *Decryption* oracle.

2. (A *new*, B *new*, C *old*): This case is impossible, as for Case 2.

3. (A *new*, B *old*, C *new*): The bad event occurs only when tag $T_1 = T_1'$ as in CASE-1. This happens with probability $\frac{q_D}{2^k}$.

4. (A *new*, B *old*, C *old*):

   (a) A$\neq \alpha$: B and C are old queries in this case and hence K, T_1 is already known to the adversary. T_2 is also fixed due to the query $Y||C^e$ to the *Decryption* oracle. Further, $\lfloor \pi(A) \rfloor_r = K \oplus T_2$ results in $T_1 = T_1'$, which is a collision of output of π(A) over k-bit value. Probability of this event is $\frac{q_D}{2^k}$ for q_D queries to the *Decryption* oracle.

   (b) A=α: This results in $T_2 = T_2^*$ due to the permutation property of π. This leads to $c_i = c_i^*$ for all i such that $1 \leq i \leq n$. If $K = K^*$, then $Y = Y^*$ which is not allowed. On the other hand, if $K \neq K^*$ and $T_1 = T_1'$, then the OAE part results in collision over k-bits. This is a kind of hash collision on outputs of OAE for different inputs. Probability of such a hash collision is $\frac{q_{\pi_A} + q_{\pi^{-1}}}{2^k}$.

5. (A *old*, B *new*, C *new*): The bad event occurs only when Tag $T_1 = T_1'$ as in CASE-1,3. This happen only with probability $\frac{q_D}{2^k}$.

6. (A *old*, B *new*, C *old*): This case is impossible. It is due the fact that if B is new, then all the subsequent inputs to π_{Dec} including C are also new.

7. (A *old*, B *old*, C *new*): Same as CASE-1,3,5.

8. (A *old*, B *old*, C *old*): The bad event can not occur in this case.

Game G7 and Game G8: Both the games are same. Game G7 and G8 both return $\perp$ when a *new query* is given to the *Decryption* oracle. In Game G8, a message M is returned only when all the input-output relations of π, which would be possible during the encryption of M, are already in $I_\pi^{\mathcal{A}}$. Game G8 iterates over all the possible pairs of (input,output) of $\pi \in I_\pi^{\mathcal{A}}$. As shown in Fig. 3.8, with help of IV_1, IV_2 and IV_3 as starting points, the decryption simulator starts checking $I_\pi^{\mathcal{A}}$ and extracts a C^f and C^e for a K such that $Y = f(C^f)$. This makes the *Decryption* oracle independent of f.

On query $Y||C^e$, the *Decryption* oracle returns a valid M only if the adversary knows the plaintext-ciphertext pair $(M, Y||C^e)$; otherwise it returns $\perp$.

$$|Pr[Exp_{G8,\mathcal{A}}] = Pr[Exp_{G7,\mathcal{A}}]|.$$

Game G8 and G9 : Both G8 and G9 act similarly. We start incremental changes in *Encryption* oracle from Game G9. In G9, K^* is chosen before Encryption query, after "find" stage once M_d is known in game. This replacement of K^* generation is shown in dash-box in G9 in Fig. 3.10. In both G8 and G9 a random K^* is used therefore,

$$|Pr[Exp_{G9,\mathcal{A}}] = Pr[Exp_{G8,\mathcal{A}}]|.$$

Game G9 and Game G10: In G9, K^* is generated randomly. In G10, K^* is computed using randomly generated values c_i^* $(1 \le i \le e)$,T_1^* and T_2^*. The value of K^* is calculated via $H^{\pi_{Enc}}(IV, c_1^*, c_2^*, \ldots, c_e^*, T_1^*) \oplus T_2^*$. Since π is an ideal permutation and T_2^* is a random value, K^* will also be random. In G10 we also mark as $Bad_K \leftarrow true$ if the adversary queries $K^*||0^{b-k}||IV_2$ to $\pi_{\mathcal{A}}$ or receives response $K^*||0^{b-k}||IV_2$ from $\pi_{\mathcal{A}}^{-1}$. In G10, Bad_K is a dummy event only. Therefore, G9 and G10 are same.

$$|Pr[Exp_{G10,\mathcal{A}}] = Pr[Exp_{G9,\mathcal{A}}]|.$$

Game $\boxed{\textbf{G9}}$ **and** $\boxed{\textbf{G10}}$ **and G11:** Initialize $I_\pi = I_{Enc} = I_\pi^{\mathcal{A}} = \emptyset$, $IV_1 = 0^r$ $IV_2 = 0^c, IV_3 = IV_2 \oplus 1$, $\mathcal{L}_c = \{IV_2, IV_3\}$,

After Find Stage (AFS): $\boxed{K^* \xleftarrow{\$} \{0,1\}^r}$

$C^*||T_1^*||T_2^* \xleftarrow{\$} \{0,1\}^{e*r+2k}$, Let $C^* = c_1^*||\ldots||c_n^*||c_{n+1}^*||\ldots||c_e^*$, where $|c^*_{[1,\ldots,e]}| = r, |T^*_{[1,2]}| = k$

$K^*||* = \pi_{Enc}(\pi_{Enc}(\ldots(\pi_{Enc}(\pi_{Enc}(c_1^*||0^c \oplus IV) \oplus c_2^*||0^c) \oplus c_3^*||0^c)\ldots \oplus c_e^*||0^c) \oplus T_1^*||0^{b-r}) \oplus T_2^*||0^{b-r}$

For G9 and G10

On Encryption-Query M_d

1. $pad(M) = m_1||m_2||\ldots||m_e$, where $|m_i| = r$ $\forall 1 \le i \le e$
2. $x = x \oplus K^*||0^{r-k}$
3. **for** $i = 1 \to e$ **do**
   - $(x||w) = \pi(x||w)$
   - $x = x \oplus m_i$
   - $c_i^* = x$
4. $(x||w) = \pi(x||w)$; $T_1^* = \lfloor x \rfloor_k$
5. $x = IV_1$ and $w = IV_3$
6. **for** $i = 1 \to e$ **do**
   - $x = x \oplus c_i^*$
   - $(x||w) = \pi(x||w)$
7. $x = x \oplus T_1^*||0^{r-k}$
8. $(x||w) = \pi(x||w)$
9. $T_2^* = \lfloor x \rfloor_k \oplus K^*$
10. $C^{f*} = c_1^*||c_2^*||\ldots||c_n^*||T_1^*||T_2^*$
11. $C^{e*} = c_{n+1}^*||\ldots||c_e^*$
12. $Y^* = f(C^{f*})$
13. Return: $Y^*||C^{e*}$

For G11

On Encryption-Query M_d

1. $C^f = c_1^*||c_2^*||\ldots||c_n^*||T_1^*||T_2^*$
2. $C^e = c_{n+1}^*||\ldots||c_e^*$
3. $Y = f(C^f)$
4. Return: $Y||C^e$

On $\pi_{\mathcal{A}}$**-Query** m, where $m \in \{0,1\}^b$

1. if $m = K^*||0^{r-k}||IV_2$ then $Bad_K \leftarrow true$
2. $v = \pi(m)$
3. $I_\pi^{\mathcal{A}} = I_\pi^{\mathcal{A}} \bigcup \{(m, v)\}$
4. return v;

On $\pi_{\mathcal{A}}^{-1}$**-Query** v, where $v \in \{0,1\}^b$

1. $m = \pi^{-1}(v)$
2. if $m = K^*||0^{r-k}||IV_2$ then $Bad_K \leftarrow true$
3. $I_\pi^{\mathcal{A}} = I_\pi^{\mathcal{A}} \bigcup \{(m, v)\}$
4. return v;

rest of oracles same as G8

Figure 3.10: Game G9,10,11 of $\mathcal{F}$-SpAEP

Game G10 and Game G11: In the game G11, K^* is generated in same way as in G10. In *Encryption* oracle, π is a ideal permutation which results in random $c_i^*(1 \le i \le e)$. Therefore, in G11, the values of c_i for all i are replaced by random values c_i^*. Similarly T_1 is replaced with T_1^*. Due to initial random K^*, $T_2 = T_2^*$. Both games G10 and G11 will behave the same way until 'Bad_K'. The Bad_K event occurs when the adversary queries $K^*||0^{b-k}||IV_2$ to $\pi_{\mathcal{A}}$ or receives response $K^*||0^{b-k}||IV_2$ from $\pi_{\mathcal{A}^{-1}}$. In G11, K^* is calculated using C^*, unlike C using the K^* as in G10. In G10, relation between $c_1, c_2, \ldots, c_e$ is generated by K^*. However, relation between $c_1^*, c_2^*, \ldots, c_e^*$ does not exist in G11. This gap in the relation can be exploited by the adversary if adversary queries $K^*||0^{b-k}||IV_2$ to $\pi_{\mathcal{A}}$ or receives response $K^*||0^{b-k}||IV_2$ from $\pi_{\mathcal{A}^{-1}}$.

$$|\Pr[Exp_{G11,\mathcal{A}}] - Pr[Exp_{G10,\mathcal{A}}]| = \Pr[Bad_K].$$

Game G12: Initialize $I_\pi = I_{Enc} = I_\pi^{\mathcal{A}} = \emptyset$, $IV_1 = 0^r$ $IV_2 = 0^c, IV_3 = IV_2 \oplus 1$, $\mathcal{L}_c = \{IV_2, IV_3\}$, Given $Y^*||C^{e*}$ for some random $C^{f*} = f^{-1}(Y^*)$, Let $C^* = C^{f*}||C^{e*}$, where $C^{f*} = c_1^*||c_2^*||\ldots||c_n^*||T_1^*||T_2^*, C^{e*} = c_{n+1}^*||\ldots||c_e^*$ $K^*||* = \pi_{Enc}(\pi_{Enc}(\ldots(\pi_{Enc}(\pi_{Enc}(c_1^*||0^c \oplus IV) \oplus c_2^*||0^c) \oplus c_3^*||0^c) \ldots \oplus c_e^*||0^c) \oplus T_1^*||0^{b-r}) \oplus T_2^*||0^{b-r}$

On Encryption-Query $M_d(d \xleftarrow{\$} \{0,1\})$

1. Return: $y||C^e = f(C^{f*}||T_1||T_2)||C^{e*}$;

On $\pi_{\mathcal{A}}$-Query m

1. $if\ m = K^*||IV_2\ then$
   $bad_K \leftarrow true$
2. $v = \pi(m)$
3. $I_\pi^{\mathcal{A}} = I_\pi^{\mathcal{A}} \bigcup\{(m, v)\}$
4. return v;

On $\pi_{\mathcal{A}}^{-1}$-Query v

1. $m = \pi^{-1}(v)$
2. $if\ m = K^*||IV_2\ then$
   $bad_K \leftarrow true$
3. $I_\pi^{\mathcal{A}} = I_\pi^{\mathcal{A}} \bigcup\{(m, v)\}$
4. return v;

Red Color text shown hidden/unknown in Game

Figure 3.11: Game G12 of $\mathcal{F}$-SpAEP

Adversary $\mathcal{B}_{\mathcal{A}}$: Given $y||C^e \xleftarrow{\$} \{0,1\}^{e*r+2k}$, where $y \in \{0,1\}^{\ell}$. Find C^f such that $f^{-1}(C^f) = y$.

Adversary $\mathcal{A}$: Initialize $I_\pi = I_{Enc} = I_\pi^{\mathcal{A}} = \emptyset$, $IV_1 = 0^r$ $IV_2 = 0^c, IV_3 = IV_2 \oplus 1$, $\mathcal{L}_c = \{IV_2, IV_3\}$,

On Encryption-Query M_d by $\mathcal{A}$

1. Return: $y||C^e$;

Finalization: if $\exists K, T_1, T_2$ such that $((K||0^{r-k} \oplus IV_1)||IV_2, v_{1_1}||v_{2_1})$, $(x_e||w_e, T_1||P||v_{2_{e+1}})$, $((z_{1_{e+1}} \oplus T_1||0^{0^{r-k}})||u_{2_{e+1}}, (T_2 \oplus K)||P||z_{2_{e+2}}) \in I_\pi^{\mathcal{A}}$ and $f(c_1||\ldots||c_n||T_1||T_2) = y$, then return $C^f = (c_1||\ldots||c_n||T_1||T_2)$, where $P \in \{0,1\}^{r-k}$

Figure 3.12: Adversary $\mathcal{B}_{\mathcal{A}}$

Game G11 and Game G12: Game G12 is the final game of adversary $\mathcal{A}$. From G11, a random Y is the output of *Encryption* oracle for random C^f and complete $C = C^f||C^e$ is unknown to adversary independent of M_d. If a random C is given to $\mathcal{A}$ in G12, then K^* will be unknown to the adversary. Bad_K event in G11 is same as Bad_K in G12.

$$|Pr[Exp_{G12,\mathcal{A}}] = Pr[Exp_{G11,\mathcal{A}}]|$$

If a random C is given to the $\mathcal{A}$ in G12, then K^* will be unknown to the adversary and C will be independent of d of M_d Therefore, $Pr[Exp_{G12,\mathcal{A}}] = \frac{1}{2}$.

Given a target ciphertext Y, Adversary $\mathcal{B}_{\mathcal{A}}$ uses $\mathcal{A}$ as a black box, while $\mathcal{A}$ uses G12.

An abstract description of adversary $\mathcal{B}$ is given in Fig 3.12. The probability of Bad_K is as follows.

$$\begin{aligned}
\Pr[Bad_K] &= \Pr[K^*||0^{r-k}||IV_2 \text{ is queried to } (\pi_{\mathcal{A}} \text{ or } \pi_{\mathcal{A}}^{-1})] \\
&= \Pr[(K^*||0^{r-k}||IV_2 \text{ is queried to } (\pi_{\mathcal{A}} \text{ or } \pi_{\mathcal{A}}^{-1})) \wedge (I_{Enc} \subset I_\pi^{\mathcal{A}})] \\
&\quad + \Pr[(K^*||0^{r-k}||IV_2 \text{ is queried to } (\pi_{\mathcal{A}} \text{ or } \pi_{\mathcal{A}}^{-1})) \wedge (I_{Enc} \not\subset I_\pi^{\mathcal{A}})].
\end{aligned}$$

$(I_{Enc} \subset I_\pi^{\mathcal{A}})$ implies that all the input-output relations of π_{Enc} are also known to the adversary $\mathcal{A}$ via set $I_\pi^{\mathcal{A}}$. Therefore $\mathcal{A}$ knows all c_i^* for $1 \le i \le e$ and T_1^*. Moreover, the adversary $\mathcal{A}$ learns T_2^* if it is allowed to query $K^*||0^{b-k}||IV_2$.

Given $Y||C^e$, if $K^*||0^{b-k}||IV_2$ is queried to ($\pi_{\mathcal{A}}$ or $\pi_{\mathcal{A}}^{-1}$), then it reveals C completely. Therefore,

$$\Pr[Bad_K] = Adv_f^{owtp}(\mathcal{B}_{\mathcal{A}}) + \Pr[(K^*||0^{r-k}||IV_2 \text{ is queried to } (\pi_{\mathcal{A}} \text{ or } \pi_{\mathcal{A}}^{-1})) \wedge (I_{Enc} \not\subset I_\pi^{\mathcal{A}})].$$

$I_{Enc} \not\subset I_\pi^{\mathcal{A}}$ implies that one of the inputs to $H^{\pi_{Enc}}()$ is unknown to the adversary $\mathcal{A}$. It results in unknown output value from $H^{\pi_{Enc}}()$. Since T_2 is already random therefore K^* remains unknown and random to $\mathcal{A}$. Therefore, $K^*||0^{r-k}||IV_2$ query to $\pi_{\mathcal{A}}$ is equivalent to random guessing of K^*.

$$\Pr[Bad_K]{=}Adv_f^{owtp}(\mathcal{B}_{\mathcal{A}}){+}\frac{(q_{\pi_{\mathcal{A}}}+q_{\pi^{-1}})}{min(2^k,2^c)}.$$

From Definition 3 and Section 2.1, if f is one-way trapdoor permutation from a family of trapdoor one-way permutations $\mathcal{F}$, then $Adv_{\mathcal{F}}^{ow}(\mathcal{B}_{\mathcal{A}}) \le negl(k)$.

The time and space complexities mentioned in Equation 3.1 are easy to verify.

This completes the proof of Theorem 1. □

3.4 Conclusion

We presented a new variant, SpAEP, of OAEP using Sponge constructions that does not require hash output of arbitrary length, whereas all previous OAEP based encryption proven secure in random-oracle model require one or more hash output of arbitrary length. Versatility of Sponge construction helps us to reduce number of round function as compared to previous OAEP-type schemes (OAEP, OAEP+, OAEP-3R, OAEP-4X, etc.,). Ability of handling long messages enables the use of SpAEP with any trapdoor one-way permutation as hybrid encryption.

3.4.1 Subsequent scope

Subsequent scope of this work can be derived from following questions. First question arise from development of OAEP type schemes. After being used with fixed length trapdoor one-way cryptosystem, OAEP type padding schemes were heavily used as KEM instantiations in hybrid encryption. Security properties and requirements for KEM are slightly different from a conventional IND-CCA secure asymmetric encryption scheme. We would like to see if $\mathcal{F}$-SpAEP can be converted into a KEM or not; and can perform better with more features when compared to existing hybrid encryption schemes. This interest of SpAEP as KEM conversion is addressed in upcoming chapter 4.

Second question arises from the limitations of the $\mathcal{F}$-SpAEP scheme. During contribution and discussion, we faced primarily two limitations of SpAEP. One, SpAEP applies to trapdoor one-way permutations only. This limitation ruled out the usage of probabilistic one-way cryptosystem like El-Gamal cryptosystem. Secondly, SpAEP is completely two-pass scheme during decryption, unlike encryption. In addition to this, during encryption, single pass feature of SpAEP is not benefiting the asymmetric encryption $(SpAEP - E_f(\cdot))$. This happens because computation of asymmetric part (f) is dependent on last output of SpAEP computation, which is dependent on entire message input. These two limitations add another question to be answered and will be discussed in forthcoming chapter 5.

Chapter 4

Sponge based KEM with partial message recovery

Contents

In this chapter, we present a *Sponge based key encapsulation mechanism with partial message recovery* (SpRKEM) using $\mathcal{F}$-SpAEP which is introduced in the previous chapter.

First, we briefly describe hybrid encryption and the benefits of *key encapsulation mechanism with partial message recovery* (RKEM) over conventional KEM. Next, we introduce first RKEM, proposed by Bjørstad et al. [25], giving a description and explaining its role as hybrid encryption. We discuss benefits of SpRKEM compared to RKEM. These benefits also provide a way to eliminate the decryption overhead of SpAEP scheme that we had discussed at the end of the previous chapter. Finally, we describe Sponge based RKEM (SpRKEM)along with its security proof followed by a description and security proof of a hybrid encryption scheme using SpRKEM and DEM. At the end of this chapter, we draw out the conclusion and derive subsequent scope of work.

4.1 Key encapsulation mechanism with partial message recovery: RKEM

In the hybrid paradigm, an asymmetric key encapsulation mechanism (KEM) combines with a symmetric data encapsulation mechanism (DEM). Traditionally, KEM is a probabilistic algorithm that produces a random symmetric key and an asymmetric encryption of that key as the key encapsulation. DEM is a deterministic algorithm that takes a symmetric key, generated by KEM, and encrypts the message under that key. Final ciphertext results from concatenation of key encapsulation and encryption of message. This traditional hybrid paradigm suffers from high ciphertext overhead (difference between plaintext and ciphertext length) equal to the size of key-encapsulation.

In 2007, Bjørstad et al. [25] introduced KEM with partial message input/recovery (RKEM). This RKEM helps in significant reduction of ciphertext overhead in hybrid constructions. In [25], for constructing RKEMs, primary focus is given to those constructions which are randomness recoverable, as happens in OAEP-type schemes. Therefore, [25] mentioned the use of RSA-OAEP in RKEMs as an example. This signifies that the OAEP-type schemes are good candidates for constructing RKEMs in practice. Therefore, any improvement in OAEP-type schemes will also help in the efficient instantiation of the RKEMs.

First, we discuss the *key encapsulation mechanism with partial message input/recovery* (RKEM) introduced by Bjørstad et al. [25]. The following description of RKEM is similar to the definition provided in [25] due to its generic nature.

4.1.1 Description

Key encapsulation mechanism with partial message recovery(RKEM) comprises a set of three algorithm RKEM = (RKEM.Gen, RKEM.Encap, RKEM.Decap) run as follows:

1. RKEM.Gen generates a (public, private) key pair $(\mathsf{sk}, \mathsf{pk})$ for security parameter k. As part of pk, there are two more parameters, *RKEM.mlen* and *RKEM.keylen*. The value *RKEM.mlen* specifies a finite value as size of message data that may be stored in an encapsulation. The value *RKEM.keylen* specifies symmetric key length, a fixed value proportional to the security parameter k, needed for symmetric cipher DEM.

2. RKEM.Encap takes input message $M0$ of length at most *RKEM.mlen* along with public key pk. The algorithm outputs a symmetric key R, of length *RKEM.keylen* and an encapsulation Y.

3. RKEM.Decap, takes input Y and outputs either $\perp$ or a pair $(R, M0)$.

A primary difference between RKEM and KEM is usage of partial message M0 as input in KEM. Conceptually, in KEM key encapsulation is computed on an internal random value, out of that internal random value a part is taken as symmetric key R. Bjørstad et al. realize that using a random value as full base for key encapsulation is not required, a major part of that internal random value can be replaced by partial message which helps in decreasing of ciphertext overhead.

4.1.2 Security notion

In terms of security, RKEM essentially needs to satisfy IND-CCA security of regular KEM in which the adversary tries to distinguish whether a given key is the one embedded in a specified encapsulation. In addition to IND-CCA security, RKEM also requires an additional security requirement known as *Real-or-Random* (ROR-CCA) for confidentiality of the message used as input and in this security definition an adversary is unable to tell a valid encryption of a message from a random ciphertext.

A short experiment of ROR-CCA for RKEM is as follows:

Experiment: $Exp^{\mathsf{ROR-CCA}}_{RKEM,\mathcal{A}}(k)$

1. $(\mathsf{pk}, \mathsf{sk}) \xleftarrow{\$} \mathrm{RKEM.Gen}(k)$
2. $(M_0, s) \leftarrow \mathcal{A}_1^{RKEM.Decap(\cdot)}$**;**
3. $M_1 \xleftarrow{\$} \{0,1\}^{|M_0|}; d \xleftarrow{\$} 0, 1; (R^*, Y^*) \leftarrow RKEM.Encap(M_d);$**.**
4. $d' \leftarrow \mathcal{A}_2^{RKEM.Decap(\cdot)}(R^*, Y^*, s);$
5. return d'**;**

$\mathcal{A}$ wins the game if $d = d'$. The advantage of $\mathcal{A}$ is given as

$$Adv^{\mathsf{ROR-CCA}}_{RKEM}(\mathcal{A}) = \mid Pr[\mathcal{A}\ wins] - \tfrac{1}{2} \mid$$

4.1.3 Constructing RKEMs

Bjørstad et al. [25] show that any IND-CCA secure public key encryption scheme can safely converted into in a RKEM.

In particular, IND-CCA secure public key encryption schemes which use a random variable K during encryption and can recover the K during decryption are preferred for constructing RKEMs.

These RKEMs use a CCA secure public key encryption $\mathcal{E}$ and a hash function H during encapsulation. On receiving a partial message $M0$, the RKEM computes encapsulation $Y = \mathcal{E}(\mathsf{pk}, M0, K)$ for some randomness K and also computes a symmetric key $R = H(M0||K)$. The RKEM outputs (R, Y) where key R is used by the DEM to encrypt the rest of the message $M1$. During decapsulation, the RKEM takes Y as an input and outputs either $\perp$ or the pair $(M0, K)$ using the decryption algorithm $\mathcal{D}(\mathsf{sk}, Y)$. Using $(M0, K)$, RKEM proceeds to compute $R = H(M0||K)$ and finally returns $(M0, R)$. This R is used further to decrypt the rest of the ciphertext through DEM. A abstract graphical representation is shown in Figure 4.1a.

4.1.4 Benefits of having a KEM/DEM version of SpAEP

- SpAEP is unified system where all internal functions work together and there is no separate session key generation in between. On the other hand, the KEM/DEM paradigm has two components: KEM and DEM, which provides options to use different customized module as KEM and DEM depending upon system requirement.

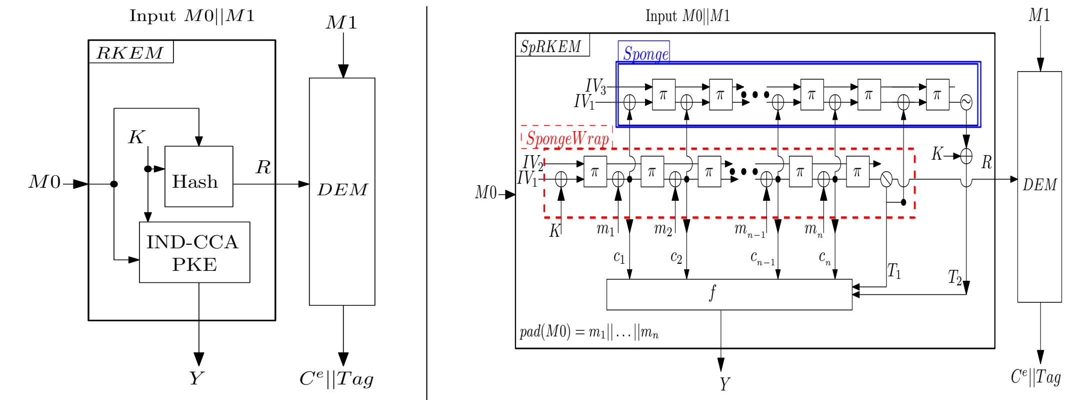
- In SpAEP, decryption cannot be performed before the complete ciphertext is received by the recipient. On the other hand in the KEM/DEM paradigm, a receiver can start decapsulation of the key using KEM and then can begin performing decryption of ciphertext using DEM. This helps us in mitigating the limitation of decryption overhead from SpAEP.

4.2 Sponge based key encapsulation mechanism with partial message recovery: SpRKEM

In this section, we describe SpRKEM based on $\mathcal{F}$-SpAEP; a graphical representation is shown in Figure 4.1b.

4.2.1 Description

SpRKEM use the same structure as that of the RKEM [25] except that it avoids using an extra H by exploiting the Sponge structure of SpAEP. In SpRKEM, $\mathcal{F}$-SpAEP takes a partial message $M0$ and outputs Y and symmetric key R using some randomness K. During decapsulation, SpRKEM takes Y as the input and returns either $(M0, R)$ or $\perp$. This symmetric key R is actually an extended part of SpongeWrap tag T_1 output, in other words, the SpongeWrap part of SpAEP now outputs $c_1||\ldots||c_n||T_1||R$.



(a) RKEM/DEM general view (b) SpRKEM/DEM

Figure 4.1: SpRKEM: $\mathcal{F}$-SpAEP as RKEM version. RKEM requires a hash function and IND-CCA PKE, whereas SpRKEM requires only $\mathcal{F}$-SpAEP as a IND-CCAPKE.

Why extract R like this? This idea of extracting key R comes from the method used in RKEM, where symmetric key R is dependent on $M0$ and randomness K through hash function H. In SpRKEM, R is also dependent on $M0$ and used randomness K through SpongeWrap which is also performing as a hash function for input $M0$ and K.

Algorithm 3: Encapsulation: **SpRKEM.Encap**(M)=(Y, R) where $|M| \leq \ell - 2k - 2$

Initialization: $IV_1 = 0^r$, $IV_2 = 0^c$, $IV_3 = IV_2 \oplus 1$, $w = IV_2, x = IV_1$
Random Nonce: $K \xleftarrow{\$} \{0,1\}^k$
$pad(M) = m_1||m_2||\ldots||m_n$, where $|m_i| = r\ \forall 1 \leq i \leq n$
$x = x \oplus K^*||0^{r-k}$
for $i = 1 \rightarrow n$ **do**
    $(x||w) = \pi(x||w)$
    $x = x \oplus m_i$
    $c_i = x$
$(x||w) = \pi(x||w)$; $T_1 = \lfloor x \rfloor_k$; $R = \lceil x \rceil_k$
$x = IV_1$ and $w = IV_3$
for $i = 1 \rightarrow n$ **do**
    $x = x \oplus c_i$
    $(x||w) = \pi(x||w)$
$x = x \oplus T_1||0^{r-k}$
$(x||w) = \pi(x||w)$
$T_2 = \lfloor x \rfloor_k \oplus K$
$C^f = c_1||c_2||\ldots||c_n||T_1||T_2$
$Y = f(C^f)$
Return: Y and R

Algorithm 4: Decapsulation: **SpRKEM.Decap** y, where $|y| = \ell$

Initialization: $IV_1 = 0^r$, $IV_2 = 0^c$, $IV_3 = IV_2 \oplus 1$, $w = IV_3, x = IV_1$
$c_1||c_2||\ldots||c_n||T_1||T_2 = f^{-1}(Y)$
for $i = 1 \rightarrow n$ **do**
    $x = x \oplus c_i$
    $(x||w) = \pi(x||w)$
$x = x \oplus T_1||0^{r-k}$
$(x||w) = \pi(x||w)$; $K = \lfloor x \rfloor_k \oplus T_2$; $R = \lceil x \rceil_k$
$x = K||0^{r-k}$; $w = IV_2$
for $i = 1 \rightarrow n$ **do**
    $(x||w) = \pi(x||w)$
    $m_i = x \oplus c_i$
    $x = c_i$
$(x||w) = \pi(x||w)$; $T_1' = \lfloor x \rfloor_k$
if $T_1 = T_1'$ **then**
    if $\exists\ M\ s.t.\ M = unpad(m_1||\ldots||m_e)$ **then**
        Return: M and R
    else
        Return: $\perp$
else
    $\perp$.

SpRKEM.Gen algorithm is similar to RKEM.Gen algorithm. Algorithms SpRKEM.Encap and SpRKEM.Decap are shown in Algorithms 3 and 4 respectively.

4.2.2 Security of SpRKEM

Theorem 2. *If the underlying trapdoor permutation f is one way, then SpRKEM is IND-CCA secure in the Ideal permutation model. The advantage of adversary $\mathcal{A}$ for the IND-CCA attack is*

$$Adv^{\mathsf{IND-CCA}}_{SpRKEM}(\mathcal{A}) \leq \frac{(q-1)q}{2^{b+1}} + \frac{q(q+1)}{2^c} + \frac{5q_D}{2^k} + \frac{q_{\pi_\mathcal{A}}}{2^k} + Adv^{ow}_{\mathcal{F}}(\mathcal{B}_\mathcal{A}\ Succeeds) + \frac{q_{\pi_\mathcal{A}}}{min(2^k, 2^c)},$$

where q is the total number of (π and π^{-1}) queries, $q_{\pi_\mathcal{A}}$ is the number of π and π^{-1} queries by $\mathcal{A}$, q_D is the number of decryption queries and b, c, k are the same as defined earlier, $\mathcal{B}$ is an adversary that finds the complete input C^f of trapdoor one way permutation f given $Y \xleftarrow{\$} \{0,1\}^\ell$ such that $Y = f(C^f)$, without having knowledge of f^{-1}. Adversary $\mathcal{B}$ uses $\mathcal{A}$ as a subroutine internally. $Adv^{ow}_{\mathcal{F}}(\mathcal{B}_\mathcal{A}\ Succeeds)$ is the success advantage that a particular adversary $\mathcal{B}$ has in breaking the trapdoor one-way permutation f of Family $\mathcal{F}$.

Proof. The proof follows the proof for $\mathcal{F}$-SpAEP. The initial Game G0 is shown in Fig. 4.2, while the rest of the games are similar to the Games of $\mathcal{F}$-SpAEP with some obvious steps of symmetric key R generation but acting as dummy steps.

Game G0: Initialize $I_\pi = I_{Enc} = I_{\mathcal{A}} = \emptyset$, $IV_1 = 0^r$ $IV_2 = 0^c, IV_3 = IV_2 \oplus 1$

On Encapsulation-Query M, where $|M| \leq \ell - 2r$

1. Random Nonce: $K^* \xleftarrow{\$} \{0,1\}^k$
2. $pad(M) = m_1||m_2||\ldots||m_n$
3. $x = x \oplus K^*||0^{r-k}$
4. **for** $i = 1 \to n$ **do**
   - $(x||w) = \pi(x||w)$
   - $x = x \oplus m_i$
   - $c_i = x$
5. $(x||w) = \pi(x||w)$; $T_1 = \lfloor x \rfloor_k$; $R_0 = \lceil x \rceil_k$
6. $x = IV_1$ and $w = IV_3$
7. **for** $i = 1 \to n$ **do**
   - $x = x \oplus c_i$
   - $(x||w) = \pi(x||w)$
8. $x = x \oplus T_1||0^{r-k}$
9. $(x||w) = \pi(x||w)$
10. $T_2 = \lfloor x \rfloor_k \oplus K^*$
11. $C^f = c_1||c_2||\ldots||c_n||T_1||T_2$
12. $Y = f(C^f)$
13. $R_1 \xleftarrow{\$} \{0,1\}^k$
14. $d \xleftarrow{\$} \{0,1\}$
15. Return: Y and R_d

On Decapsulation-Query Y

1. $C^f = c_1||c_2||\ldots||c_n||T_1||T_2 = f^{-1}(Y)$
2. $C = c_1||c_2||\ldots||c_n||T_1||T_2$,
3. **for** $i = 1 \to n$ **do**
   - $x = x \oplus c_i$
   - $(x||w) = \pi(x||w)$
4. $x = x \oplus T_1||0^{r-k}$
5. $(x||w) = \pi(x||w)$; $K = \lfloor x \rfloor_k \oplus T_2$; $R = \lceil x \rceil_k$
6. $x = K||0^{r-k}$; $w = IV_2$
7. **for** $i = 1 \to n$ **do**
   - $(x||w) = \pi(x||w)$
   - $m_i = x \oplus c_i$
   - $x = c_i$
8. $(x||w) = \pi(x||w)$; $T_1' = \lfloor x \rfloor_k$
9. **if** $T_1 = T_1'$ **then**
   - **if** $\exists$ M *s.t.* $M = unpad(m_1||\ldots||m_e)$ **then**
     - Return: M and R
   - **else**
     - Return: **Invalid**

   else
   - Invalid.

Rest Same as G0 of SpAEP

Figure 4.2: Game G0 for IND-CCA security of SpRKEM

□

Theorem 3. *If the underlying trapdoor permutation f is one way, then SpRKEM is ROR-CCA secure in the Ideal permutation model. The advantage of adversary $\mathcal{A}$ for ROR-CCA attack is*

$$Adv_{SpRKEM}^{ROR-CCA}(\mathcal{A}) \leq \frac{(q-1)q}{2^{b+1}} + \frac{q(q+1)}{2^c} + \frac{5q_D}{2^k} + \frac{q_{\pi_\mathcal{A}}}{2^k} + Adv_{\mathcal{F}}^{ow}(\mathcal{B}_\mathcal{A}\ Succeeds) + \frac{q_{\pi_\mathcal{A}}}{min(2^k, 2^c)},$$

where q is the total number of (π and π^{-1}) queries, q_π and $q_{\pi^{-1}}$ are the number of π and π^{-1} queries, $q_{\pi_\mathcal{A}}$ is the number of π queries by $\mathcal{A}$, q_D is the number of *decryption queries and* b, c, k *are the same as defined earlier,* $\mathcal{B}$ *is an adversary that finds the complete input* C^f *of trapdoor one way permutation* f *given* $y \xleftarrow{\$} \{0,1\}^\ell$ *such that* $y = f(C^f)$*, without having knowledge of* f^{-1}*. Adversary* $\mathcal{B}$ *uses* $\mathcal{A}$ *as a subroutine internally.* $Adv_f^{ow}(\mathcal{B}_\mathcal{A}$ *Succeeds) is the success advantage that a particular adversary* $\mathcal{B}$ *has in breaking the trapdoor one-way permutation* f*.*

Proof. The proof exactly follows the one for $\mathcal{F}$-SpAEP. Initial Game G0 is shown in Fig. 4.3, rest of the games exactly follow the games of $\mathcal{F}$-SpAEP with some obvious steps of symmetric key K generation acting as dummy steps.

Game G0: Initialize $I_\pi = I_{Enc} = I_{Dec} = I_\mathcal{A} = \emptyset$, $f : \{0,1\}^\ell \to \{0,1\}^\ell$, $IV_1 = 0^r$ $IV_2 = 0^c, IV_3 = IV_2 \oplus 1$

On Encryption-Query M_0, where $|M| \leq \ell - 2k$

1. Random Nonce: $K^* \xleftarrow{\$} \{0,1\}^k$
2. $M_1 \xleftarrow{\$} \{0,1\}^{|M_0|}$
3. $d \xleftarrow{\$} \{0,1\}$
4. $pad(M_d) = m_1||m_2||\ldots||m_n$, where $|m_i| = b_i$ $\forall 1 \leq i \leq n$
5. $x = x \oplus K^*||0^{r-k}$
6. **for** $i = 1 \to n$ **do**
   - $(x||w) = \pi(x||w)$
   - $x = x \oplus m_i$
   - $c_i = x$
7. $(x||w) = \pi(x||w)$; $T_1 = \lfloor x \rfloor_k$; $R = \lceil x \rceil_k$
8. $x = IV_1$ and $w = IV_3$
9. **for** $i = 1 \to n$ **do**
   - $x = x \oplus c_i$
   - $(x||w) = \pi(x||w)$
10. $x = x \oplus T_1||0^{r-k}$
11. $(x||w) = \pi(x||w)$
12. $T_2 = \lfloor x \rfloor_k \oplus K^*$
13. $C^f = c_1||c_2||\ldots||c_n||T_1||T_2$
14. $y = f(C^f)$
15. Return: y and R

On Decryption-Query y

1. $C^f = c_1||c_2||\ldots||c_n||T_1||T_2 = f^{-1}(y)$
2. $C = c_1||c_2||\ldots||c_n||T_1||T_2$, where $|c_i| = b_i, |T_1| = |T_2| = r$ for $1 \leq i \leq n$
3. **for** $i = 1 \to n$ **do**
   - $x = x \oplus c_i$
   - $(x||w) = \pi(x||w)$
4. $x = x \oplus T_1||0^{r-k}$
5. $(x||w) = \pi(x||w)$; $K = \lfloor x \rfloor_k \oplus T_2$; $R = \lceil x \rceil_k$
6. $x = K||0^{r-k}$; $w = IV_2$
7. **for** $i = 1 \to n$ **do**
   - $(x||w) = \pi(x||w)$
   - $m_i = x \oplus c_i$
   - $x = c_i$
8. $(x||w) = \pi(x||w)$; $T_1' = \lfloor x \rfloor_k$
9. **if** $T_1 = T_1'$ **then**
   - **if** $\exists$ M *s.t.* $M = unpad(m_1||\ldots||m_e)$ **then**
     - Return:M and K
   - **else**
     - Return: **Invalid**

   else
   - Invalid.

Figure 4.3: Game G0 of ROR-CCA security of SpRKEM

□

The IND-CCA security notion is related to the advantage of the adversary to determine whether a given key is indeed produced after the encapsulation or produced randomly. On the other hand, the ROR-CCA security is related to quantifying the advantage of an adversary to distinguish an encapsulated message from a random encapsulated message.

4.3 Security of Hybrid PKE scheme using IND-CCA and ROR-CCA secure SpRKEM and an IND-PA and INT-CTXT secure DEM

4.3.1 Description

(**PKE=SpRKEM+DEM**) Given a SpRKEM=(SpRKEM.Gen, SpRKEM.Encap, SpRKEM.Decap) and DEM= (DEM.Enc, DEM.Dec), as per definitions 4.2 and 2.4.2 respectively, the symmetric key output of SpRKEM is the same as the input of the DEM. A public key encryption scheme PKE=(PKE.Gen, PKE.Enc, PKE.Dec) with security parameter k is described as follows:

- The key generation algorithm PKE.Gen uses SpRKEM.Gen to generate a pair of public and private keys (f, f^{-1}) having input-output length ℓ.
- The encryption algorithm PKE.Enc is as follows:
  1. The input message M split into $M0$ and $M1$, i.e. $M = M0||M1$, where $|M0| = (\ell - 2k - 2 - 1)$ and $|M1| = |M| - |M0|$ if $|M| \geq (\ell - 2k - 2 - 1)$ else $|M0| = |M|$ *and* $|M1| = \varnothing$
  2. **If** $|M1| \neq \varnothing$ **then**
     Compute $(R, Y) = SpRKEM.Encap(M0||1)$
     Compute $\chi = DEM.Enc_R(M1)$
     else
     Compute $(K, y) = SpRKEM.Encap(M0||0)$
     $\chi = \emptyset$
  3. Output $C = (R, \chi)$
- The decryption algorithm PKE.Dec is as follows:

1. Parse input $C = (Y||\chi)$, where $|Y| = \ell$.
2. Compute $(R, M') = SpRKEM.Decap(y)$.
3. **If** (R, M')=$\perp$ **then** return $\perp$.
4. **If** $(M' = M0||1)$ and $\chi \neq \emptyset$ **then**
   Compute $M1 = DEM.Dec_R(\chi)$.
   else
   If $(M' = M0||0)$ and $\chi = \emptyset$ **then** M1=$\varnothing$
   else Return $\perp$
5. **If** $M1 \neq \perp$ **then** Return $M0||M1$
   else Return $\perp$

4.3.2 Security

Theorem 4. *Given an IND-CCA and ROR-CCA secure SpRKEM and an IND-PA and INT-CTXT secure DEM, the Hybrid PKE composition is IND-CCA secure. The advantage of an adversary $\mathcal{A}$ for IND-CCA attack for PKE scheme is given by*

$$\begin{aligned} Adv_{PKE}^{\textsf{IND-CCA}}(\mathcal{A}) \leq 2 \cdot Adv_{SpRKEM}^{\textsf{IND-CCA}}(\mathcal{B}_1) + q_D \cdot Adv_{DEM}^{INT-CTXT}(\mathcal{B}_2) \\ + 2 \cdot Adv_{SpRKEM}^{\textsf{ROR-CCA}}(\mathcal{B}_3) + Adv_{DEM}^{\textsf{IND-}PA}(\mathcal{B}_4), \end{aligned}$$

where $\mathcal{B}_1$,$\mathcal{B}_2$,$\mathcal{B}_3$ and $\mathcal{B}_4$ are polynomial time adversaries, and q_D is the upper bound on the number of queries made by $\mathcal{A}$ to PKE.Dec.

Proof. This proof exactly follows the proof of Theorem 1 in [25].
Let Game G0 be the standard IND-CCAgame for a PKE and Adversary $\mathcal{A}$ is the adversary against the PKE system. In Game i, let G_i denote the event that $d = d'$. Hence,

$$Adv_{PKE}^{\textsf{IND-CCA}}(\mathcal{A}) =| Pr[G_0] - \tfrac{1}{2}$$

Let Game 1 be same as Game 0 except that if adversary asks challenger to decrypt a ciphertext(Y^*, χ), where Y^* is equal to encapsulation part of challenge, then it uses the key R^* output by encapsulation SpRKEM when it decrypt χ. The algorithms used in DEM and SpRKEM still work perfectly, and this query case will be just a conceptual difference. Therefore,

$$Pr[G_0] = Pr[G_1].$$

Game 2 remains same as Game 1 , except for computation of χ^* of challenge ciphertext, in which a random key R' is used instead of R^*. We known that, there exists a adversary $\mathcal{B}_1$, which can distinguish Game 2 from Game 1 as adversary against the IND-CCA property of SpRKEM. Therefore, we have

$$\mid Pr[G_1] - Pr[G_2] \mid \leq 2 \cdot Adv^{\mathsf{IND-CCA}}_{SpRKEM}(\mathcal{B}_1).$$

Game 3 remains same as Game 2 until the adversary asks the challenger to decrypt a ciphertext (Y^*, χ). Due to Y^*, while decrypting χ by DEM using random $R^* = R'$, it simply rejects ciphertext. We know that there exists an adversary $\mathcal{B}_2$, which can distinguish Game 3 from Game 2 as an adversary against INT-CTXT property of DEM. Therefore,

$$\mid Pr[G_2] - Pr[G_3] \mid \leq q_D \cdot Adv^{INT-CTXT}_{DEM}(\mathcal{B}_2).$$

In Game 4, we start making changes in computation of encapsulation part. We replace the $M0$ from a random string of equal length during encapsulation SpRKEM, but second part of encapsulation using DEM remains same. We know that there exists an adversary $\mathcal{B}_3$, which can distinguish between Game 4 and Game 3 by acting as an adversary against the ROR-CCA property of SpRKEM. Therefore,

$$\mid Pr[G_3] - Pr[G_4] \mid \leq 2 \cdot Adv^{\mathsf{ROR-CCA}}_{SpRKEM}(\mathcal{B}_3).$$

In Game 4, first part of ciphertext computation using SpRKEM is completely random and independent of any input message and due to generation of a random key $R^* = R'$, second part of ciphertext using DEM is also random. Therefore, the adversary in Game 4 is an adversary which is attacking the IND-PA property of DEM. Therefore, we have

$$\mid Pr[G_4] - \tfrac{1}{2} \mid \leq Adv^{\mathsf{IND}-PA}_{DEM}(\mathcal{B}_4).$$

Combining all the game equalities we have,

$$\begin{aligned} Adv^{\mathsf{IND-CCA}}_{PKE}(\mathcal{A}) \leq & 2 \cdot Adv^{\mathsf{IND-CCA}}_{SpRKEM}(\mathcal{B}_1) + q_D \cdot Adv^{INT-CTXT}_{DEM}(\mathcal{B}_2) \\ & + 2 \cdot Adv^{\mathsf{ROR-CCA}}_{SpRKEM}(\mathcal{B}_3) + Adv^{\mathsf{IND}-PA}_{DEM}(\mathcal{B}_4). \end{aligned}$$

$\square$

4.4 Conclusion

Bjørstad et al. [25] suggested that any CCA-secure public key encryption scheme that supports randomness recovery, like OAEP-based schemes, are suitable for using $\mathcal{E}_{\mathsf{pk}}$ as part of RKEM in practice. Usage of H in RKEM is needed for generation of symmetric key R. In SpRKEM, SpongeWrap as the OAE inherits the property of H used in the RKEM to generate a random symmetric key R. Therefore, since $\mathcal{F}$-SpAEP has already been proven CCA secure and is randomness recoverable also, it can directly replace both $\mathcal{E}_{pk}$ and H required in the RKEM of [25], which leads to the scheme we have introduced as SpRKEM.

4.4.1 Subsequent scope

We observe that conversion of $\mathcal{F}$-SpAEP to SpRKEM+DEM is same as converting from monolithic construction to hybrid construction. We found that SpRKEM+DEM reduces the decryption overhead of $\mathcal{F}$-SpAEP scheme. In SpRKEM+DEM, encryption and decryption become single pass because SpRKEM computations are independent of input-output of DEM during encryption/decryption. It would be interesting to discuss how we can modify $\mathcal{F}$-SpAEP such that its encryption and decryption process also become single pass like in hybrid encryption mode.

We would also like to remind an another limitation from last chapter about usage of trapdoor one-way permutation only, which is also a limitation in proposed SpRKEM.

Both of these limitations, decryption overhead and incompatibility with probabilistic one-way asymmetric cryptosystem, will be discussed in details in upcoming Chapter 5 along with a solution to remove these limitations.

Chapter 5

Sponge based padding for CCA-secure Asymmetric encryption from trapdoor one-way functions

Contents

In this chapter, we discuss a modified variant of SpAEP which we denote as SpPad. SpPad is compatible with probabilistic one-way cryptosystem also, unlike SpAEP which is compatible only with trapdoor one-way permutations (deterministic one-way cryptosystem). Other than just trapdoor one-way permutation (e.g. RSA), it is easy to describe other one-way secure cryptosystems from any trapdoor problem. Further more, such trapdoor problems are not so rare (Diffie-Hellman [51], Elliptic Curves [64], McEliee [75], NTRU [60], etc.).

First, we explain the reasons of usage restriction of OAEP-type schemes with trapdoor one-way permutation only and existing solutions to remove these restrictions. Next we provide features and benefits of SpPad, a modified SpAEP, compared to other existing schemes. Further, we describe and explain the modifications on SpAEP according to existing solutions to achieve SpPad. We provide description of SpPad and then security proof of SpPad when used with probabilistic one-way cryptosystem. Proposed SpPad is shown to be a combination of positive outcomes from SpRKEM and SpAEP while removing some of their existing limitations. We provide conclusion and subsequent scope of work at the end of the chapter.

5.1 Motivation

5.1.1 Limitation to Trapdoor one-way permutation

From previous chapters, we can see considerable improvements have taken place in OAEP type schemes. Some of these are lowering the security assumption and having support for long messages with the help of symmetric encryption schemes. However, OAEP-type schemes are usable only with deterministic one-way cryptosystem (like RSA), and not with probabilistic one-way cryptosystem (like ElGamal). A prime reason for this limitation is the probabilistic nature of such one-way cryptosystem. Conceptually, in probabilistic one-way cryptosystem, encryption function takes an input M and uses a new random coin g to produce output Y. Any change in the randomness g will produce a different Y' even for the same input M. The presence of g implies that there is not just one Y but a set of possibilities $\mathcal{Y}$ such that the same M is obtained for all $Y \in \mathcal{Y}$ under decryption. This allows a favorable condition for an adversary to chose any $Y \in \mathcal{Y}$ as a query to decryption function resulting in the same M.

Any public key encryption scheme that is probabilistic and homomorphic will allow the user to re-randomize a given ciphertext by using the homomorphic property with a ciphertext. The El-Gamal encryption scheme is susceptible to such attacks, where an adversary can change a given Y to Y' without knowing M, and then can receive that M by using the decryption function. This type of attack is also known as the re-randomization attack.

Most of the probabilistic one-way cryptosystem generate the random coin g within the encryption function instead of receiving it as an external parameter. Further, these schemes do not return g along with M during decryption of the ciphertext. This hidden value of g creates an uncertain mapping between Y and M when randomness g or secret key sk are not revealed. This uncertain mapping causes difficulty in having an efficient decryption simulation without the secret key, which is an important step in the security proof for IND-CCA security.

5.1.2 Candidate solutions

The first generic solution to prevent the re-randomization attack was proposed by Fujisaki and Okamoto [55] by means of the FO-transform. In brief, a random g is generated which is explicitly dependent on M and some other random parameters. This g can be recovered during decryption and subsequently the relation between M and Y can be checked by re-encrypting M with g. Fujisaki and Okamoto were able to propose a CCA-secure encryption scheme from any OW cryptosystem, but their scheme needs a re-encryption mechanism during decryption and suffers from high ciphertext-overhead like other hybrid schemes.

Another solution for this problem was proposed by Okamoto and Pointcheval in [83]. Instead of handling g explicitly, they introduced two changes. Firstly, a hash value of Y and M is generated which serves as a checksum. This hash value becomes a part of the final ciphertext along with Y. Introduction of the hash value in the ciphertext disallows an adversary to submit another Y' as part of ciphertext to the decryption oracle as the adversary will be unable to generate the hash of Y' without knowing M, where M is protected under one-way property of the cryptosystem using some g. Secondly, they elevate the security property of underlying one-way cryptosystem (Pe) from OW to OW-PCA. In OW-PCA, there exists a public oracle $\mathcal{O}^{PC}$ which can output 1 if given (M, Y) pair is a right pair and 0 if it is not. Availability of this oracle helps in simulating decryption oracle without using secret key sk for providing IND-CCA security proof. As shown in [82], ElGamal [57] encryption achieves OW-PCA security under GDH assumption, whereas only CDH assumption is enough for one-wayness of ElGamal. This modification makes the system more practical by avoiding re-encryption mechanism during decryption. Moreover, for deterministic one-way cryptosystem security notion of OW is equivalent to $\mathsf{OW\text{-}PCA}$. Thus, if a PKE works securely with a $\mathsf{OW\text{-}PCA}$ secure Pe, then that PKE also works securely with a deterministic OW cryptosystem.

Both of the schemes discussed above proposed a hybrid encryption where KEM part does not use OAEP-type structure. Hybrid encryption schemes suffer from a high ciphertext overhead except for RKEM-DEM [25] or some specific KEM constructions, but RKEM-DEM requires stronger security assumptions. In light of existing limitation of $\mathcal{F}$-SpAEP and benefits of SpRKEM over RKEM, we investigate how integration of existing solution, to remove usage restriction of type of one-way cryptosystem, will work with $\mathcal{F}$-SpAEP. We also investigate whether Sponge based padding can provide some enhancement in features not only to $\mathcal{F}$-SpAEP but also to the scheme REACT proposed in [83].

Another motivation of our work is to achieve "on the fly" encryption and decryption property, with lower ciphertext-overhead along with support of probabilistic one-way cryptosystem. We achieve this by merging the security assumption given in [83] into $\mathcal{F}-$SpAEP with appropriate modifications.

5.2 Contribution

This work proposes a generic framework that converts OW-PCA asymmetric primitive (Pe) into a CCA-secure and efficient PKE. Apart from Pe, the framework requires a permutation, which operates in a iterated fashion like Sponge function [20]. This permutation behave as an ideal permutation. Encryption scheme constructed using our framework is denoted by SpPad–Pe.

Security of SpPad–Pe is proven in ideal permutation model, unlike FO-transform [55] and REACT [83] which used the RO model. Security results of the scheme are similar to the FO-transform and REACT, but along with lower ciphertext overhead and the addition of "on the fly" encryption and decryption. Note that "on the fly" computation has significant applications in memory constrained devices and streaming applications in networks.

SpPad–Pe achieves lower ciphertext overhead compared to FO-transform and REACT. Let us denote $\mathsf{co}_{\mathsf{Pe}}$ to be ciphertext overheads of Pe having input length ℓ. Let $k < \ell$ be the length of random strings used in the scheme, then the ciphertext overhead of SpPad–Pe is $(\mathsf{co}_{\mathsf{Pe}} + 2k)$. The ciphertext overhead of FO-transform is

Generic Schemes	Asymmetric Encryption	Model	Re-Encryption	Ciphertext overhead	# of other functions	on the fly Enc	on the fly Dec
FO [55]	OW	RO	Yes	$\ell + co_{pe}$	2Hash + 1 SE	No	Yes
REACT [83]	OW-PCA	RO	NO	$\ell + co_{pe} + k$	2hash + 1 SE	Yes	Yes
SpAEP (Chap.3)	OW (only permutation)	Ideal permutation (P)	NO	$2k$	P based 1 Hash + 1 SE	Yes	NO
SpPad–Pe (This chapter)	OW-PCA	Ideal permutation (P)	NO	$co_{\mathsf{Pe}} + 2k$	P based 1 Hash + 1 SE	Yes	Yes

Table 5.1: Generic CCA transformations:
"$\ell + co_{pe}$" is output length and ℓ is input length of asymmetric encryption
"k" is security parameter, "SE" is Symmetric encryption
"OW" is one-wayness and "PCA"" is plaintext checking attack

($\mathsf{co}_{\mathsf{Pe}} + \ell$) and that of REACT is ($\mathsf{co}_{\mathsf{Pe}} + \ell + k$) due to hybrid nature of schemes.

The computation time of SpPad–Pe during encryption is lower than FO-trasnform. Let t_{asym} and t_{sym} be computation time for asymmetric primitive and symmetric cipher, and $t^{\ell}_{sym}(< t_{sym})$ be the computation time of symmetric cipher to output ℓ bits. The resulting computation time of FO scheme and $\mathcal{F}$-SpAEP will be $t_{asym} + t_{sym}$. On the other hand, SpPad–Pe will have $\mathtt{max}^1(t^{\ell}_{sym} + t_{asym},\, t_{sym})$ computation time. This decrease in the computation time in SpPad–Pe compared to FO-transform might appear to be very small. However, it is significant for very long messages. The decryption time of SpPad–Pe is similar to the scheme REACT, where no re-encryption is required during decryption. Compared to $\mathcal{F}$-SpAEP, SpPad–Pe has lower decryption time due to the early recovery of randomness without performing a full pass over the complete ciphertext.

SpPad–Pe also provides streaming capability, which is a useful feature, specially in broadcast systems. Once the asymmetric part is processed (encrypted/ decrypted) in SpPad–Pe the data can be streamed using the symmetric encryption/decryption. Although REACT also provides streaming during encryption as well as decryption, the ciphertext overhead of SpPad–Pe is lower than REACT.

Our work is a direct extension of the scheme $\mathcal{F}$-SpAEP, from Chapter 3, by removing the restriction of using only trapdoor one-way permutation as $\mathcal{F}$. Now any trapdoor one-way cryptosystem (Pe) can be used. The restriction of using only trapdoor one-way permutation as Pe is overcome by having OW-PCA assumption on Pe, by following results of [40, 83]. Our work also results in decreasing computation overhead during decryption and encryption which enable

[1] max function return a the maximal value amongst the parameters.

us to provide "on-the-fly" computation. An early recovery of used randomness for symmetric decryption helps in decreasing the decryption overhead.

A summary of results in provided in Table 5.1 along with comparison against most representative CCA secure generic PKEs. The last line in the table shows our SpPad–Pe construction. Note that it compares favorably with other constructions.

Finally, it is interesting to observe that the framework we describe in this chapter is quite different from a regular hybrid encryption. Recall that the hybrid encryption uses two systems, namely KEM and DEM with a clear delineation between the two. In our framework, the two overlap.

5.3 Sponge based padding with one-way cryptosystem

5.3.1 Description

Sponge based padding ($SpPad^{\pi}$) is based on an iterated ideal permutation $\pi : \{0,1\}^{(b=r+c)} \rightarrow \{0,1\}^b$ with an fixed initial value IV. $SpPad^{\pi}$ or simply denote as $SpPad$ uses the functionality of SpongeWrap and *Sponge* function together in some dependent way, under different initial values to keep domain separation. Initial value used for SpongeWrap is $IV_1||IV_2$ where $IV_1 = 0^r$ and $IV_2 = 0^c$. *Sponge* uses $IV_1||IV_3$ as initial value where $IV_3 = IV_2 \oplus 1$. For some fixed value k and ℓ, where $\ell = n \cdot r$ for $n > 0$ and $\ell \geq r > c > k$.

SpPad work always with an another function say F. This function F takes input X and outputs Y, where $|X| = \ell$. *SpPad* working with F is denoted as *SpPad-F*.

If we denote a one-way cryptosystem as Pe, then SpPad with trapdoor one-way cryptosystem is denote as SpPad–Pe. The building blocks for SpPad–Pe are:

1. an asymmetric encryption scheme Pe: (Gen, Enc, Dec) of minimum input message size ℓ as described in Section 2.1.

2. an ideal permutation $\pi : \{0,1\}^{b=r+c} \rightarrow \{0,1\}^b$.

For simple understanding, we assume $\ell = n * r$ for some positive integer $n \geq 1$. *SpPad*–Pe is defined as a triplet of the following probabilistic polynomial-time (PPT) algorithms: $\langle SpPKE.Gen, SpPKE.Enc, SpPKE.Dec \rangle$.

- *SpPKE.Gen* produces a private/public key pair$(\mathsf{pk}, \mathsf{sk})$ using $\mathsf{Gen}(1^k)$.

- *SpPKE.Enc* encrypts a message M under pk, and produces a cryptogram $\chi = Y||K_h||C^e||T$.

  This algorithm generates a random K, random coin g if needed, and takes input message M. Using *SpongeWrap* on input M and K, it generates partial output $C||T_k$, where $T_k = T \oplus K$. The C split into C^f and C^e as $C = C^f||C^e$ where $|C^f| = \ell$. Enc takes C^f as input and under pk (and random coin g, if required) outputs Y. *Sponge* takes $C^f||Y$ as input and outputs h. Here, *Sponge* uses $pad(C^f||Y, r, \varnothing)$. This leads to final output of *SpPKE.Enc* as $\chi = Y||K_h||C^e||T_k$, where $K_h = K \oplus h$.
  Encryption pseudo-code is shown in Algorithm 5.

- *SpPKE.Dec* recovers a plaintext M from a ciphertext χ under sk. On input $\chi = Y||K_h||C^e||T_k$, Dec outputs C^f under sk for input Y. *Sponge* then uses input $C^f||Y$ to generate value h. K is calculated as $h \oplus K_h$. Using K and C^f
  $C^e||T_k$ SpongeWrap finally outputs M or $\perp$.
  Pseudo-code of decryption is shown in Algorithm 6 .

  Graphical representation of *SpPad* with a trapdoor one-way function (Pe) is provided in Figure 5.1.

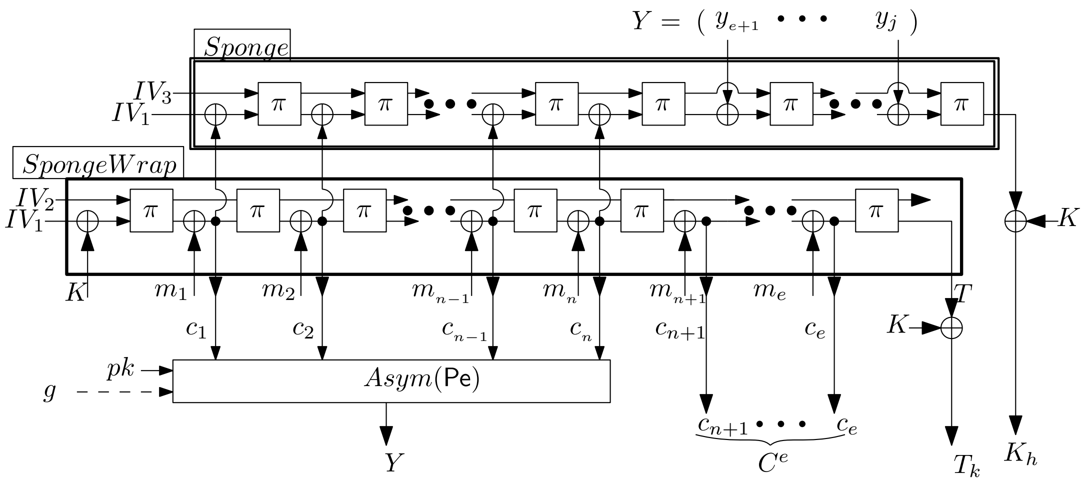


Figure 5.1: Sponge based padding for Trapdoor one-way functions.
Dashed line in figure represents optional g used when random coins are required.

Algorithm 5: Encryption: $SpPKE.Enc(M) = Y||K_h||C^e||T_k$

Initialization: $x = IV_1 = 0^r$, $w = IV_2 = 0^c$, $IV_3 = IV_2 \oplus 1$,
Random Nonce: $K \xleftarrow{\$} \{0,1\}^k$
Random coins: $g \xleftarrow{\$}$ `COINS`
$pad(M, r, \ell) = m_1||m_2||\ldots||m_e$, where $|m_i| = r\ \forall 1 \leq i \leq e$
$x = x \oplus K||0^{r-k}$
for $i = 1 \to e$ **do**
  $(x||w) = \pi(x||w)$
  $x = x \oplus m_i$
  $c_i = x$
$(x||w) = \pi(x||w)$; $T = \lfloor x \rfloor_k$
$C^f = c_1||c_2||\ldots||c_n$; $C^e = c_{n+1}||\ldots||c_e$
$Y = \mathsf{Enc}_{\mathsf{pk}}(C^f; g)$
$y_1||\ldots y_j = pad(C^f||y, r, \varnothing)$
$x = IV_1$ and $w = IV_3$
for $i = 1 \to j$ **do**
  $x = x \oplus y_i$
  $(x||w) = \pi(x||w)$
$h = \lfloor x \rfloor_k$;$K_h = h \oplus K$; $T_k = T \oplus K$
Return: $Y||K_h||C^e||T_k$

Algorithm 6: Decryption: $SpPKE.Dec(Y||K_h||C^e||T_k) = M\ or\ \perp$

Initialization: $x = IV_1 = 0^r$, $IV_2 = 0^c$, $w = IV_3 = IV_2 \oplus 1$
$C^f = \mathsf{Dec}_{\mathsf{sk}}(Y)$;
$y_1||\ldots||y_j = pad(C^f||Y, r)$
for $i = 1 \to j$ **do**
  $x = x \oplus y_i$
  $(x||w) = \pi(x||w)$
$h = \lfloor x \rfloor_k$;$K = h \oplus K_h$; $T = T_k \oplus K$
$x = IV_1 \oplus K||0^{r-k}$; $w = IV_2$
$c_1||c_2||\ldots||c_n = C^f$; $c_{n+1}||\ldots||c_e = C^e$
for $i = 1 \to e$ **do**
  $(x||w) = \pi(x||w)$
  $m_i = x \oplus c_i$
  $x = c_i$
$(x||w) = \pi(x||w)$; $T' = \lfloor x \rfloor_k$
if $T = T'$ **then**
  Return:$unpad(m_1||\ldots||m_n)$
else
  Return: $\perp$.

5.3.2 Structural difference between SpAEP and SpPad

Some notable differences between SpAEP of chapter 3 and SpPad are in inputs to *Sponge* part and to Pe.

Conceptually, in SpAEP the entire output of SpongeWrap part is input to *Sponge* part and recoverable randomness K is bound, in last step, with output of *Sponge*. This binding of K in last step causes delay in recovery of K during decryption. In SpPad, length of this input part to *Sponge* is shortened. Only a part of output of SpongeWrap C^f, which is input to Pe, is taken as input in *Sponge*. This enables the early recovery of randomness K during decryption without passing over the entire ciphertext.

Regarding Pe input, SpAEP takes T and K_h also as a part of input to Pe along with C^f. We realize that for security purposes, it is not necessary to protect all three (T, K_h, C^f) under one-wayness of Pe, only protecting C^f is enough. Because if C^f is known to adversary then calculating K from K_h using π is trivial. Therefore, in SpPad only C^f gets protection under one-wayness of Pe and K is automatically protected. Using this follow-up protection T also get protected once $T_k = T \oplus K$ is computed. Therefore, K_h and T_k used directly as part of final ciphertext. This also helps in executing Enc independently from T unlike SpAEP, which is dependent on entire plaintext.

These two modifications in SpPad help in decreasing double pass overhead during decryption and make encryption independent from entire input compared to SpAEP.

5.3.3 CCA security of SpPad–Pe

Now we are ready to present a security proof of CCA security of *SpPad*–Pe. We assume that H, G and F are independent random oracles. Nature of proof and bound calculation will be followed in similar manner like of $\mathcal{F} - SpAEP$. The experiment of adversary $\mathcal{A}$ for *SpPad*–Pe or simply $SpPKE$ is as follows:

Experiment: $Exp^{ind-cca}_{SpPad\text{–}\mathsf{Pe},\mathcal{A}}(k)$
1. $(\mathsf{pk}, \mathsf{sk}) \xleftarrow{\$} \mathsf{Gen}(1^k)$;
2. $(M_0, M_1, s) \leftarrow \mathcal{A}_1^{\pi(\cdot), SpPKE.Dec(\cdot)}(\mathsf{pk})$;
3. $d \xleftarrow{\$} \{0,1\}$
4. $\chi^* \leftarrow SpPKE.Enc(M_d)$; ...one time encryption query
5. $d' \leftarrow \mathcal{A}_2^{\pi(\cdot), SpPKE.Dec(\cdot)}(\mathsf{pk}, \chi^*, s)$;
6. return d';

Theorem 5. *Given a* OW-PCA *asymmetric encryption primitive* Pe:(Gen, Enc, Dec), *an ideal permutation* $\pi : \{0,1\}^{b=r+c} \to \{0,1\}^b$, *then the construction of SpPad–*Pe *defined in Section 5.3.1 is* IND-CCA *secure. The success probability of an adversary* $\mathcal{A}$ *is*

$$\Pr[Exp^{ind-cca}_{SpPad\text{-}\mathsf{Pe},\mathcal{A}}(k) = d] \leq \frac{1}{2} + \frac{(q-1)q}{2^{b+1}} + \frac{q(q+1)}{2^c} + \frac{4q_d}{2^k} + Adv^{OW-PCA}_{\mathsf{Pe}}(\mathcal{B}_{\mathcal{A}}\ Succeeds) + \frac{q_{\pi_{\mathcal{A}}}}{2^k},$$

where q_d is number of queries to the decryption oracle, q is total number of queries to the π oracle and $q_{\pi_{\mathcal{A}}}$ is number of queries to the π oracle by adversary. $\mathcal{B}$ is an adversary that finds the complete input X of Pe *given y and* pk *such that $y = \mathsf{Enc}_{\mathsf{pk}}(X; g)$, for some randomness g if present, without knowing* sk*. $Adv^{OW-PCA}_{\mathsf{Pe}}(\mathcal{B}_{\mathcal{A}})$ is the success advantage that a particular adversary $\mathcal{B}$ has in breaking* OW-PCA *security of* Pe.

Proof. We will use game based playing technique [15, 16]. We start from the original CCA game as defined in Section 2.2. $Exp_{SpPKE,\mathcal{A}}$ Or $Exp^{ind-cca}_{SpPad\text{-}\mathsf{Pe},\mathcal{A}}(k) = d$ denote the event that $\mathcal{A}$ outputs $d' = d$ where $d \xleftarrow{\$} \{0, 1\}$. We want to show that $|Pr[Exp_{SpPKE,\mathcal{A}}]|=\frac{1}{2} + negl(k)$. We slightly change $SpPKE$ into a sequence G0, G1, ..., G12 such that:
$Pr[Exp_{SpPKE,\mathcal{A}}]= Pr[Exp_{G0,\mathcal{A}}]$
$Pr[Exp_{G(i-1),\mathcal{A}}]=Pr[Exp_{Gi,\mathcal{A}}]+negl(r)\ \forall 1 \leq i \leq 11$
$Pr[Exp_{G12,\mathcal{A}}]= \frac{1}{2}$

- *Encryption* ($SpPKE.Enc$), *Decryption*($SpPKE.Dec$): perform Encryption and Decryption,
- π, π^{-1}: public invertible permutation and its inverse,
- π_{Enc}: permutation π calls by encryption,
- $\pi_{\mathcal{A}}$, $\pi^{-1}_{\mathcal{A}}$: permutation π, π^{-1} calls by adversary $\mathcal{A}$.

Encryption, *Decryption*, $\pi_{\mathcal{A}}$ and $\pi^{-1}_{\mathcal{A}}$ are public oracles, which are also accessible to the adversary. In each game, the following sets are maintained: I_π by π and π^{-1}, I_{enc} by π_{Enc} and $I^{\mathcal{A}}_{\pi}$ by $\pi_{\mathcal{A}}$ and $\pi^{-1}_{\mathcal{A}}$ to store input-output relations.

Another set $\mathcal{L}_c : \{g : g \in \{0, 1\}^c\}$ is also maintained internally by π and π^{-1} for storing capacity bits. The set $\mathcal{L}_c$ is initialized to $\{IV_2, IV_3\}$ because IV_2 is the capacity part of the input to first π of OAE part and IV_3 is the capacity part of the input to the first π of *Hash* part. The set Y is updated on every call to π. Precisely, two c-bit values are appended to $\mathcal{L}_c$ on each π call. These two values are the capacity bits of the inputs and output of π.

Note that $q = q_\pi + q_{\pi^{-1}}$, $q_\pi = q_{\pi_\mathcal{A}} + q_{\pi_{Enc}}$ and q_d= number of decryption queries.

Challenge ciphertext χ^* has $C^*, Y^*, C^{f*}, C^{e*}, K^*, h^*, R^*_h, g^*, T^*$ and T^*_k as corresponding internal values during computation of challenge query.

In each of the games G0, G1, G2, G3, G4, G5 we make small incremental changes in the permutation to make it ideal permutation. In games G6, G7, we make changes in the *Decryption* oracle and make it independent of sk. Finally, in games G8, G9, G10, G11, G12 we make changes in *Encryption* oracle along with some changes in $\pi_\mathcal{A}$ oracle to achieve that d of M_d is independent of all previous queries. We represent the *Sponge* part of SpPad as a function $H^\pi(j_1, j_2, j_3..., j_i, j_{i+1})$ whose output J is such that

$$J||* = \pi_{Enc}\Big(\pi_{Enc}(\dots(\pi_{Enc}(\pi_{Enc}(j_2||0^{b-r} \oplus j_1) \oplus j_3||0^{b-r}) \oplus j_4||0^{b-r})\dots \oplus j_{i-1}||0^{b-r}\Big) \oplus j_i||0^{b-r}) \oplus j_{i+1}||0^{b-r}$$

where π is b-bit permutation, $j_1 \in \{0,1\}^b$, $(j_2, j_3, \dots j_{i+1}) \in \{0,1\}^r$, $J \in \{0,1\}^k$ and $* \in \{0,1\}^{b-k}$.

Game G0: This game perfectly simulates the $SpPad$–Pe.

$$\Pr[Exp_{SpPKE,\mathcal{A}}] = \Pr[Exp_{G0,\mathcal{A}}].$$

Game G0 to G5: Followed exactly same as in Fig. 3.4 3.5 3.6. This gives us following bound $\frac{(q-1)q}{2^{b+1}} + \frac{q(q+1)}{2^c}$ between G0 to G5.

Game G5 and Game G6: Both the games are same. In Game G6 only a dummy operation, shown as dash-box, of $flag \leftarrow new$ is added in the *Decryption* oracle to denote a new query. The query is new in the sense that neither the query nor any part of the query during internal calls to π, of *Decryption* oracle, was queried earlier by the adversary. That is, query $\notin I^\mathcal{A}_\pi$. In decryption oracle there is addition of one more dummy line of bad_π as true if $T = T'$ happens for $flag = new$.

$$|Pr[Exp_{G6,\mathcal{A}}] = Pr[Exp_{G5,\mathcal{A}}]|.$$

Game G0: Initialize $I_{enc} = I_\pi = I_\pi^{\mathcal{A}} = \emptyset$, $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^k)$, $IV_1 = 0^r, IV_2 = 0^c$, $IV_3 = IV_2 \oplus 1$.

On Encryption-Query(M_d)

1. $K^* \xleftarrow{\$} \{0,1\}^k$
2. $g^* \xleftarrow{\$} \texttt{COINS}$
3. $m_1||m_2||\ldots||m_e = M$
4. $x = IV_1 \oplus K||0^{r-k}$, $w = IV_2$
5. **for** $i = 1 \rightarrow n \rightarrow e$ **do**
   - $(x||w) = \pi_{enc}(x||w)$
   - $x = x \oplus m_i$
   - $c_i^* = x$
6. $(x||w) = \pi_{enc}(x||w)$; $T^* = \lfloor x \rfloor_k$
7. $C^{f*} = c_1^*||c_2^*||\ldots||c_n^*$; $C^{e*} = c_{n+1}^*||\ldots||c_e^*$
8. $Y^* = \mathsf{Enc}_{\mathsf{pk}}(C^{f*}; g^*)$
9. $y_1^*||\ldots y_j^* = C^{f*}||Y^*$; $x = IV_1$ and $w = IV_3$
10. **for** $i = 1 \rightarrow j$ **do**
    - $x = x \oplus y_i$
    - $(x||w) = \pi_{enc}(x||w)$
11. $h^* = \lfloor x \rfloor_k$;$K_h^* = h^* \oplus K^*$; $T_k^* = T^* \oplus K^*$
12. Return: $Y^*||K_h^*||C^{e*}||T^*$

On Decryption-Query $\chi = Y||K_h||C^e||T$

1. $C^f = \mathsf{Dec}_{\mathsf{sk}}(Y)$; $x = IV_1$ and $w = IV_3$
2. $y_1||\ldots||y_j = C^f||Y$
3. **for** $i = 1 \rightarrow j$ **do**
   - $x = x \oplus y_i$
   - $(x||w) = \pi(x||w)$
4. $h = \lfloor x \rfloor_k$;$K = h \oplus K_h$; $T = T_k \oplus K$
5. $x = IV_1 \oplus K||0^{r-k}$; $w = IV_2$
6. $c_1||c_2||\ldots||c_n = C^f$; $c_{n+1}||\ldots||c_e = C^e$
7. **for** $i = 1 \rightarrow n$ **do**
   - $(x||w) = \pi(x||w)$
   - $m_i = x \oplus c_i$
   - $x = c_i$
8. $(x||w) = \pi(x||w)$; $T' = \lfloor x \rfloor_k$
9. **if** $T == T'$ **then**
   - Return:$unpad(m_1||\ldots||m_e)$

   else
   - Return: $\perp$.

On π-Query m

1. if $(m, v) \in I_\pi$ then return v
2. $v \xleftarrow{\$} \{0,1\}^b$
3. if $\exists$ m' s.t $(m', v) \in I_\pi$, then $v \xleftarrow{\$} \{0,1\}^b \setminus \{v : (*, v) \in I_\pi\}$, where $* \in \{0,1\}^b$
4. $I_\pi = I_\pi \bigcup \{(m, v)\}$
5. return v;

On π^{-1}-Query v,

1. if $(m, v) \in I_\pi$ then return m
2. $m \xleftarrow{\$} \{0,1\}^b$
3. if $\exists$ v' s.t $(m, v') \in I_\pi$, then $m \xleftarrow{\$} \{0,1\}^b \setminus \{m : (m, *) \in I_\pi\}$, where $* \in \{0,1\}^b$
4. $I_\pi = I_\pi \bigcup \{(m, v)\}$
5. return m;

On $\pi_{\mathcal{A}}$-Query m

1. $v = \pi(m)$
2. $I_\pi^{\mathcal{A}} = I_\pi^{\mathcal{A}} \bigcup \{(m, v)\}$
3. return v;

On $\pi_{\mathcal{A}}^{-1}$-Query v

1. $m = \pi^{-1}(v)$
2. $I_\pi^{\mathcal{A}} = I_\pi^{\mathcal{A}} \bigcup \{(m, v)\}$
3. return v;

On π_{enc}-Query m

1. $v = \pi(m)$
2. $I_{enc} = I_{enc} \bigcup \{(m, v)\}$
3. return v;

Figure 5.2: Game G0

Game $\boxed{\textbf{G6}}$ $\boxed{\textbf{G7}}$**:** Initialize $I_{enc} = I_\pi = I_\pi^{\mathcal{A}} = \emptyset$, $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^k)$, $IV_1 = 0^r, IV_2 = 0^c$, $IV_3 = IV_2 \oplus 1$. $\mathcal{L}_c = \{IV_2, IV_3\}$. $\boxed{flag \in \{new, old\}}$.

On Decryption-Query $\chi = Y||K_h||C^e||T$

1 $C^f = \mathsf{Dec}_{\mathsf{sk}}(Y)$; $x = IV_1$ and $w = IV_3$
2 $y_1||\dots||y_j = C^f||Y$; $\boxed{flag \leftarrow old}$
3 **for** $i = 1 \to j$ **do**
  $x = x \oplus y_i$
  $\boxed{\textbf{If}\{x||w,*\} \notin I_\pi^{\mathcal{A}} \textbf{ then } flag \leftarrow new}$
  $(x||w) = \pi(x||w)$
4 $h = \lfloor x \rfloor_k$;$K = h \oplus K_h$; $T = T_k \oplus K$
5 $x = IV_1 \oplus K||0^{r-k}$; $w = IV_2$
6 $c_1||c_2||\dots||c_n = C^f$; $c_{n+1}||\dots||c_e = C^e$
7 **for** $i = 1 \to e$ **do**
  $\boxed{\textbf{If}\{x||w,*\} \notin I_\pi^{\mathcal{A}} \textbf{ then } flag \leftarrow new}$
  $(x||w) = \pi(x||w)$
  $m_i = x \oplus c_i$
  $x = c_i$
8 $\boxed{\textbf{If}\{x||w,*\} \notin I_\pi^{\mathcal{A}} \textbf{ then } flag \leftarrow new}$
9 $(x||w) = \pi(x||w)$; $T' = \lfloor x \rfloor_k$
10 **if** $T == T'$ *and* $flag == new$ **then**
  $\boxed{\textbf{bad}_\pi \leftarrow true}$
  Return: $\boxed{unpad(m_1||\dots||m_e)}$ $\boxed{\perp}$
11 **if** $T == T'$ *and* $flag == old$ **then**
  Return:$unpad(m_1||\dots||m_e)$
else
  $\perp$.

Rest of Oracles same as G5

Figure 5.3: Game G6: G6 includes dummy lines, shown in dash-box,compare to G5 along with round-box

Figure 5.4: G7: G7 includes all codes of line of G6 and also solid-box but not round-box.

Game G6 and Game G7: In G7, if $\textbf{bad}_\pi$ happens then return $\perp$. Both games G6 and G7 act similarly till $\textbf{bad}_\pi$ occurs. The event $\textbf{bad}_\pi$ occurs in *Decryption* oracle when a new query results in $T_1 = T'_1$ (mentioned in Fig. 5.4 line 10). The

$\mathbf{bad}_\pi$ event occurs with probability $\frac{4q_D}{2^k}$.

$$|Pr[Exp_{G7,\mathcal{A}}] - Pr[Exp_{G6,\mathcal{A}}]| = Pr[\mathbf{bad}] \leq \frac{6q_D}{2^k}.$$

Let $(v_1||v_2) = \pi(x||w)$, where $x,v_1 \in \{0,1\}^r$ and $w,v_2 \in \{0,1\}^c$. In decryption, an input is a *new query* to π when $((x||w),(v_1||v_2)) \notin I_\pi^{\mathcal{A}}$ and *old query* when $((x||w),(v_1||v_2)) \in I_\pi^{\mathcal{A}}$. If a *new query* $(x||w)$ is input to π during decryption, then π outputs $v_1||v_2$, where $v_2 \notin \mathcal{L}_c$. That is, v_2 is also new. Since v_2 is unseen so far, it ensures that the input to the next call of π is certainly new. Further, since v_2 is new, next input $x'||v_2$ satisfies the condition $(x'||v_2,*) \notin I_\pi^{\mathcal{A}}$, where $*$ stands for any b bit value. Therefore one *new query* makes all subsequent inputs to $\pi(\cdot)$ as new. Any *new query* to π implies that a ciphertext y queried to *Decryption* oracle has never been generated by the adversary. In Game G7, *Decryption* oracle return $\perp$(**Invalid**) whenever adversary makes such a query.

To know if a *new query* has been made in SpPad–Pe *Decryption* oracle, we consider three checkpoints, called A, B and C. A is input to last block of π in *Sponge* Part, B is input of first and C is input of last π of SpongeWrap. Next we explain the situation when a $\mathbf{bad}_\pi$ event can occur in Game G7.

In *Sponge*-part, if any input before A is new, then A is also new as explained earlier. Hence a decryption query is certainly new if A is new. In the case of checkpoints B and C, it is not possible that B is *new query* and C is *old query*. This follows from our discussion above. Therefore, we only need to check C to determine if there is a *new query* in the SpongeWrap part.

During encryption, let us denote the values at checkpoints A, B and C by $\alpha, K^*||0^{b_r-r}||IV_2$ and β respectively. Let $Y^*||K_h^*||C^{e*}||T_k^*$ be the target ciphertext and $C^* = C^{f*}||C^{e*}$ where $C^{f*} = c_1^*||\ldots||c_e^*$ and $C^{e*} = c_{e+1}^*||\ldots||c_n^*$ such that $Y^* = \mathsf{Enc}_{\mathsf{pk}}(C^{f*},*)$, $K_h^* = h^* \oplus K^*$ and $Y^* = y_1^*||\ldots y_j^*$.
The following cases cover all the possible cases for *new query*.

CASE-1 (A *new*, B *new*, C *new*): The $\mathbf{bad}_\pi$ event occurs only when tag $T = T'$ (shown in Algorithm 6 and Game G7 in Fig. 5.4)

- : C$\neq \beta$: Then $T = T'$ implies collision of the outputs of π over r-bit value. Probability of this event is $\frac{q_d}{2^k}$ for q_d queries to *Decryption* oracle
- : C$=\beta$: Then $T = T^*$ which means $c_i = c_i^*$ for all i such that $1 \leq i \leq n$ and $K = K^*$. This leads to $C = C^*$. Now ,If $A = \alpha$, this results in

$Y = Y^*$ and $K_h = K_h^*$, which is not allowed because adversary can not this query to *Decryption* oracle. If $A \neq \alpha$, then h is random and probability that $K_h \oplus h = K^*$ is $q_d/2^k$.

CASE-2 (A *new*, B *new*, C *old*): This case is impossible. It is due to the fact that if B is new, then all subsequent inputs to π including C are also new.

CASE-3 (A *new*, B *old*, C *new*): This is repetition of CASE-1.

CASE-4 (A *new*, B *old*, C *old*): B and C are old queries in this case and hence K and T is already known to the adversary along with all c_i for all i such that $1 \leq i \leq n$. K_h is also fixed due to the query $Y||K_h||C^e||T_k$ to the *Decryption* oracle.

(a) A$\neq \alpha$: Further, $\lfloor \pi(A) \rfloor_k$ is random value results in $K \oplus K_h$, which is a collision of output of π(A) over k-bit value. Probability of this event is $\frac{q_d}{2^k}$ for q_d queries to the *Decryption* oracle.

(b) A$=\alpha$: This results in $h = h^*$ due to the permutation property of π. This leads to $c_i = c_i^*$ for all i such that $1 \leq i \leq e$ and $Y = Y^*$. Now probability of $K_h \oplus K = h^*$, for unknown h^* is an random event where SpongeWrap part results in collision over k-bits. This is a kind of hash collision on outputs of SpongeWrap for different inputs. Probability of such a hash collision is $\frac{q_d}{2^k}$.

CASE-5 (A *old*, B *new*, C *new*): This is repetition of CASE-1

CASE-6 (A *old*, B *new*, C *old*): This case is impossible, as for CASE-2.

CASE-7 (A *old*, B *old*, C *new*): This is repetition of CASE-1.

CASE-8 (A *old*, B *old*, C *old*): The bad event can not occur in this case.

Game G7 and Game G8: Both the games are same. Game G7 and G8 both return $\perp$ when a *new query* is given to the *Decryption* oracle. In Game G8, a message M is returned only when all the input-output relations of π, which would be possible during the encryption of M, are already in $I_\pi^{\mathcal{A}}$. Game G8 iterates over all the possible pairs of (input,output) of $\pi \in I_\pi^{\mathcal{A}}$. This makes the *Decryption* oracle independent of $\mathsf{Dec_{sk}}$.

Game G8: Initialize $I_{enc} = I_\pi = I_\pi^{\mathcal{A}} = \emptyset$, $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^k)$, $IV_1 = 0^r, IV_2 = 0^c$, $IV_3 = IV_2 \oplus 1$. $\mathcal{L}_c = \{IV_2, IV_3\}$

On Decryption-Query $\chi = Y||K_h||C^e||T_k$

1 If $\exists$ pad(M)==$m_1||m_2||\ldots||m_e$ such that
after setting $Y = a_{n+1}||\ldots||a_j$, $u_{2_1} = IV_3$, $z_{1_1} = IV_1$
if $\{(u_{1_i}||u_{2_i}), (z_{1_i+1}||z_{2_i+1})\} \in I_\pi^{\mathcal{A}}$ for $i : 1 \rightarrow n \rightarrow j$ such that
$a_i = u_{1_i} \oplus z_{1_i}$, $u_{2_i} = z_{2_i}$ and $\mathcal{O}^{PC}(C^f, y) = 1$, where $C^f = a_1||\ldots||a_n$
then for setting $K = \lfloor z_j \rfloor_r \oplus K_h$, $C^f||C^e = c_1||\ldots c_n||c_{n+1}||\ldots||c_e$
$x_0 = K||0^{r-k} \oplus IV_1$ and $w_0 = IV_2$
if $(x_0||w_0, v_{1_1}||v_{2_1}) \in I_{\mathcal{A}}$, and
$\{(x_i||w_i), (v_{1_{i+1}}||v_{2_{i+1}})\} \in I_\pi^{\mathcal{A}}$ for $i : 1 \rightarrow n \rightarrow e$ and
$\lfloor v_{1_{e+1}} \rfloor_r == T_k \oplus K$
where $x_i = c_i = m_i \oplus v_{1_i}$,$w_i = v_{2_i}$
then **Return:** M
else **Return:** $\perp$

Rest of Oracles same as G7

Following special notations is used during Game G8 and onwards in decryption oracle:

1. During SpongeWrap part of SpPad, we represent input-output relation of π's subsequent calls for $pad(M) = m_1||\ldots||m_e$ by $(v_{1_{i+1}}||v_{2_{i+1}}) = \pi(x_i||w_i)$, where $x_i = v_{1_i} \oplus \{m_i\}$, $w_i = v_{2_i}$ $0 \leq i \leq e$, $v_{1_0} = IV_1$, $m_0 = K$, $w_0 = IV_2$, $v_{1_i}, x_i \in \{0,1\}^r$ and $v_{2_i}, w_i \in \{0,1\}^c$. Then c_i will represent $m_i \oplus v_{1_i}$, where $1 \leq i \leq e$.

2. Input-output relation of π's subsequent call during *Sponge* part of SpPad will be represented as follows: $(z_{1_i+1}||z_{2_i+1}) = \pi(u_{1_i}||u_{2_i})$, $u_{1_i} = c_i \oplus z_{1_i}$, $u_{2_i} = z_{2_i}$,where $1 \leq i \leq j$, $u_{2_1} = IV_3$, $z_{1_1} = IV_1$, $z_j = h$.

Figure 5.5: Game G8: Output of decryption oracle in G8 is same as G7 but independent from sk.

On query $Y||K_h||C^e||T_k$, the *Decryption* oracle returns a valid M only if the adversary knows the plaintext-ciphertext pair $(M, Y||K_h||C^e||T_k)$; otherwise it returns $\perp$. Plaintext-checking oracle $\mathcal{O}^{PC}$ confirms if C^f extracted from $I_\pi^{\mathcal{A}}$ is a valid pair with Y under some g or not.

$$|Pr[Exp_{G8,\mathcal{A}}] = Pr[Exp_{G7,\mathcal{A}}]|.$$

Game G8 and Game G9: We start incremental changes in *Encryption* oracle from Game G9. In Game G9, K^* is chosen before encryption query and after "find" stage. In both case K^* remain random therefore,

$$|Pr[Exp_{G9,\mathcal{A}}] = Pr[Exp_{G8,\mathcal{A}}]|.$$

Game G9 and Game G10: In G9, K^* is generated randomly. In G10, K^* is computed using the value of randomly generated C^{f*}, K_h^* and subsequently random Y^* from $\mathsf{Enc}_{\mathsf{pk}}$. The value of K^* is calculated via $H^{\pi_{Enc}}(IV_1||IV_2, y_1^*, y_2^*, \ldots, y_j^*) \oplus K_h^*$, where $y_1^*, y_2^*, \ldots, y_j^* = C^{f*}||Y^*$. Since π is an ideal permutation and K_h^* is a random value, K^* will also be random. Therefore, G9 and G10 are same.

$$Pr[Exp_{G10,\mathcal{A}}] = Pr[Exp_{G9,\mathcal{A}}].$$

Game G9 G10 :Initialize $I_{enc} = I_\pi = I_\pi^{\mathcal{A}} = \emptyset$, $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^k)$, $IV_1 = 0^r, IV_2 = 0^c$, $IV_3 = IV_2 \oplus 1$. $flag \in \{new, old\}$. $\mathcal{L}_c = \{IV_2, IV_3\}$

After Find Stage(AFS): $g^* \xleftarrow{\$}$ `COINS`; $K_h^* \xleftarrow{\$} \{0,1\}^k$; $C^{f*} \xleftarrow{\$} \{0,1\}^\ell$, $C^{e*} \xleftarrow{\$} \{0,1\}^{Clen(M_d)-\ell}$ $Y^* \xleftarrow{\$} \{0,1\}^{\ell+co_{pe}}$, $T_k^* \xleftarrow{\$} \{0,1\}^k$

$(y_1||\ldots||y_n)||(y_{n+1}||\ldots||y_j) = (C^{f*})||(Y^*)$;

$(c_1||\ldots||c_n)||(c_{n+1}||\ldots||c_e) = (C^{f*})||(C^{e*})$; $K^* \xleftarrow{\$} \{0,1\}^k$

$K^*||* = \pi_{enc}(\ldots\pi_{enc}(\pi_{enc}(y_1 \oplus IV_1||IV_3) \oplus y_2||0^c)\ldots \oplus y_j||0^c) \oplus K_h^*||0^{b-k}$

On Encryption-Query(M_d)

1. $m_1||m_2||\ldots||m_n = M$
2. $x = IV_1 \oplus K||0^{r-k}$, $w = IV_2$
3. **for** $i = 1 \to e$ **do**
   - $(x||w) = \pi_{enc}(x||w)$
   - $x = x \oplus m_i$
   - $c_i^* = x$
4. $(x||w) = \pi_{enc}(x||w)$; $T_k^* = \lfloor x \rfloor_k \oplus K^*$
5. $C^{f*} = c_1^*||c_2^*||\ldots||c_n^*$; $C^{e*} = c_{n+1}^*||\ldots||c_e^*$
6. $Y^* = E_{\mathsf{pk}}(C^{f*}; g^*)$
7. $y_1^*||\ldots y_j^* = C^{f*}||Y^*$; $x = IV_1$ and $w = IV_3$
8. **for** $i = 1 \to j$ **do**
   - $x = x \oplus y_i$
   - $(x||w) = \pi_{enc}(x||w)$
9. $h^* = \lfloor x \rfloor_k$;$K_h^* = h^* \oplus K^*$
10. Return: $Y^*||K_h^*||C^{e*}||T_k^*$

Rest of Oracles same as G8

Figure 5.6: Game G9 and G10: G9 includes some extra dummy variables, shown in dash-box, during initialization after find stage. G10 includes solid-box code during initialization in which K^* is chosen from random C^*.

Game G10 and Game G11: In the game G11, K^* is generated in same way as in G10. In *Encryption* oracle, π is an ideal permutation which results in random $c_i^*(1 \le i \le e)$. Therefore, in G11, the values of c_i^* for all i are replaced by random values c_i^* independent π. Similarly T^* output of π is replaced with random T^*. Due to initial random K^*, K_h^* is also random independent from h^* of π. Because now C^{f*} is totally random therefore Y^* is also a random string, which can be replaced by any other random string chosen independently. Both games G10 and G11 will behave the same way until 'Bad_K'. The Bad_K event occurs when the adversary queries $K^*||0^{b-k}||IV_2$ to $\pi_{\mathcal{A}}$ or receives response $K^*||0^{b-k}||IV_2$ from $\pi_{\mathcal{A}}^{-1}$. In G11, K^* is calculated using C^* and Y^*, unlike C^* using the K^* as in G10. In G10, relation between $c_1^*, c_2^*, \ldots, c_n^*$ is generated by K^*. However, relation between $c_1^*, c_2^*, \ldots, c_n^*$ does not exist in G11. This gap in the relation can be exploited by the adversary if adversary queries $K^*||0^{r-k}||IV_2$ to $\pi_{\mathcal{A}}$ or receives response $K^*||0^{r-k}||IV_2$ from $\pi_{\mathcal{A}}^{-1}$.

$$|Pr[Exp_{G11,\mathcal{A}}] - Pr[Exp_{G10,\mathcal{A}}]| = \Pr[Bad_K].$$

G11: Initialize $I_{enc} = I_\pi = I_\pi^{\mathcal{A}} = \emptyset$, $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^k)$, $IV_1 = 0^r, IV_2 = 0^c$, $IV_3 = IV_2 \oplus 1$. $flag \in \{new, old\}$. $\mathcal{L}_c = \{IV_2, IV_3\}$

After Find Stage(AFS): $g^* \xleftarrow{\$} \texttt{COINS}$; $K_h^* \xleftarrow{\$} \{0,1\}^k$; $C^{f*} \xleftarrow{\$} \{0,1\}^\ell$, $C^{e*} \xleftarrow{\$} \{0,1\}^{Clen(M_d)-\ell}$; $Y^* \xleftarrow{\$} \{0,1\}^{\ell+co_{pe}}$, $T_k^* \xleftarrow{\$} \{0,1\}^k$

$(y_1||\ldots||y_n)||(y_{n+1}||\ldots||y_j) = (C^{f*})||(Y^*)$;

$(c_1||\ldots||c_n)||(c_{n+1}||\ldots||c_e) = (C^{f*})||(C^{e*})$

$K^*||* = \pi_{enc}(\ldots\pi_{enc}(\pi_{enc}(y_1 \oplus IV_1||IV_3) \oplus y_2||0^c)\ldots \oplus y_j||0^c) \oplus K_h^*||0^{b-k}$

On Encryption-Query(M_d)

1. Return: $Y^*||K_h^*||C^{e*}||T_k^*$

On $\pi_{\mathcal{A}}$-Query m

1. If $(m = K^*||0^{b-k})$ then $Bad_K \leftarrow true$
2. $v = \pi(m)$
3. $I_{\mathcal{A}} = I_{\mathcal{A}} \bigcup \{(m, v)\}$
4. return v;

On $\pi_{\mathcal{A}}^{-1}$-Query v

1. $m = \pi^{-1}(v)$
2. If $(m = K^*||0^{b-k})$ then $Bad_K \leftarrow true$
3. $I_{\mathcal{A}} = I_{\mathcal{A}} \bigcup \{(m, v)\}$
4. return v;

Rest of Oracles same as G10

Figure 5.7: Game G11: All values of encryption oracle replaced by random variables, if adversary does not query K^* to $\pi_{\mathcal{A}}$

Game G11 and Game G12: Game G12 is the final game of adversary $\mathcal{A}$. From G11, a random Y^* is the output of *Encryption* oracle and C^{f*} of C^* is unknown to adversary independent of M_d. Therefore, if a random χ is given to the $\mathcal{A}$ in G12, then K^* will be unknown to the adversary. Bad_K event in G11 is same as $Bad1_K$ in G12.

$$|Pr[Exp_{G12,\mathcal{A}}] = Pr[Exp_{G11,\mathcal{A}}]|,$$

If a random χ is given to the $\mathcal{A}$ in G12, then K^* will be unknown to the adversary and χ will be independent of d of M_d Therefore, $Pr[Exp_{G12,\mathcal{A}}] = \frac{1}{2}$.

Given a target ciphertext Y, Adversary $\mathcal{B}_{\mathcal{A}}$ uses $\mathcal{A}$ as a black box, while $\mathcal{A}$ uses G12.

A detailed description of the games and adversary $\mathcal{B}$ is given in Fig 5.8. The probability of $Bad1_K$ is as follows.

$$\begin{aligned}
\Pr[Bad_K] &= Pr[K^*||0^{b-k}||IV_2 \text{ is queried to } (\pi_{\mathcal{A}} \text{ } or \text{ } \pi_{\mathcal{A}}^{-1})] \\
&= \Pr[(K^*||0^{b-k}||IV_2 \text{ is queried to } (\pi_{\mathcal{A}} \text{ } or \text{ } \pi_{\mathcal{A}}^{-1})) \wedge (I_{enc} \subset I_{\pi}^{\mathcal{A}})] \\
&\quad + \Pr[(K^*||0^{b-k}||IV_2 \text{ is queried to } (\pi_{\mathcal{A}} \text{ } or \text{ } \pi_{\mathcal{A}}^{-1})) \wedge (I_{enc} \not\subset I_{\pi}^{\mathcal{A}})].
\end{aligned}$$

$(I_{Enc} \subset I_{\mathcal{A}})$ implies that all the input-output relations of π_{Enc} are also known to the adversary $\mathcal{A}$ via set $I_{\mathcal{A}}$. Therefore $\mathcal{A}$ knows all c_i^* for $1 \leq i \leq n$ and h^*. Moreover, the adversary $\mathcal{A}$ learns K^* from K_h^* of challenge ciphertext.

Given $Y^*||K_h^*||C^{e*}||T_k^*$, if $K^*||0^{b-k}||IV_2$ is queried to π, then it reveals C^* completely. Therefore,

$$\begin{aligned}
\Pr[Bad_K] =&\leq Adv_{\mathsf{Pe}}^{OW-PCA}(\mathcal{B}_{\mathcal{A}} \text{ } Succeeds) + \Pr[(K^*||IV_2 \text{ is queried to } (\pi_{\mathcal{A}} \\
&\quad \text{or } \pi_{\mathcal{A}}^{-1})) \wedge (I_{Enc} \not\subset I_{\mathcal{A}})].
\end{aligned}$$

$I_{Enc} \not\subset I_{\mathcal{A}}$ implies that one of the inputs to $H^{\pi_{Enc}}()$ is unknown to the adversary $\mathcal{A}$. It results in unknown output value from $H^{\pi_{Enc}}()$. Since K_h^* is already random therefore K^* remains unknown and random to $\mathcal{A}$. Therefore, $K^*||0^{b-k}$ query to $\pi_{\mathcal{A}}$ is equivalent to random guessing of K^*.

$$\Pr[Bad1_K] \leq Adv_{\mathsf{Pe}}^{OW-PCA}(\mathcal{B}_{\mathcal{A}} \text{ } Succeeds) + \tfrac{(q_{\pi_{\mathcal{A}}})}{min(2^k, 2^c)}.$$

This completes the proof of Theorem 5. □

G12: Initialize $I_{enc} = I_\pi = I_\pi^{\mathcal{A}} = \emptyset$, $(\mathsf{pk},\mathsf{sk}) \leftarrow \mathsf{Gen}(1^k)$, $IV_1 = 0^r, IV_2 = 0^c$, $IV_3 = IV_2 \oplus 1$. $\mathcal{L}_c = \{IV_2, IV_3\}$
(AFS):$K_h^* \xleftarrow{\$} \{0,1\}^r$; $C^{e*} \xleftarrow{\$} \{0,1\}^{Clen(M_d)-\ell}$; $Y^* \xleftarrow{\$} \{0,1\}^{\ell+co_{pe}}$; $T_k^* \xleftarrow{\$} \{0,1\}^k$;
where $C^f = \mathsf{Dec}_{\mathsf{sk}}(Y)$
$K^*||* = \pi_{enc}(\ldots\pi_{enc}(\pi_{enc}(y_1 \oplus IV_1||IV_3) \oplus y_2||0^c)\ldots \oplus y_j||0^c) \oplus K_h^*||0^{b-k}$

On Encryption-Query(M_d)

1. Return: $Y^*||K_h^*||C^{e*}||T_k^*$

On $\pi_{\mathcal{A}}$-Query m

1. If $(m = K^*||0^{b-k})$ then
   $Bad_K \leftarrow true$
2. $v = \pi(m)$
3. $I_{\mathcal{A}} = I_{\mathcal{A}} \bigcup \{(m,v)\}$
4. return v;

On $\pi_{\mathcal{A}}^{-1}$-Query v

1. $m = \pi^{-1}(v)$
2. If $(m = K^*||0^{b-k})$ then
   $Bad_K \leftarrow true$
3. $I_{\mathcal{A}} = I_{\mathcal{A}} \bigcup \{(m,v)\}$
4. return v;

Rest of Oracles same as G11

Red color line shows lines which are not detectable by Adversary.

Adversary $\mathcal{B}$: Given random $Y \xleftarrow{\$} \{0,1\}^{\ell+co_{pe}}$, find C^f such that $\mathsf{Enc}_{\mathsf{pk}}(C^f;*) = Y$

Game G12 as Adversary $\mathcal{A}$: Initialize $I_{enc} = I_\pi = I_\pi^{\mathcal{A}} = \emptyset$, $(\mathsf{pk},\mathsf{sk}) \leftarrow \mathsf{Gen}(1^k)$, $IV_1 = 0^r, IV_2 = 0^c$, $IV_3 = IV_2 \oplus 1$. $\mathcal{L}_c = \{IV_2, IV_3\}$
(AFS):$K_h^* \xleftarrow{\$} \{0,1\}^r$; $C^{e*} \xleftarrow{\$} \{0,1\}^{Clen(M_d)-\ell}$; $T_k^* \xleftarrow{\$} \{0,1\}^k$;

Rest of Oracles same as G12

Finalization: if $\{(u_{1_i}||u_{2_i}),(z_{1_i+1}||z_{2_i+1})\} \in I_\pi^{\mathcal{A}}$ for $i : 1 \to n \to j$ such that $a_i = u_{1_i} \oplus z_{1_i}$, $u_{2_i} = z_{2_i}$ and $\mathcal{O}^{PC}(C^f, Y) = 1$, where $C^f = a_1||\ldots||a_n$, $Y = a_{n+1}||\ldots||a_j$, $u_{2_1} = IV_3$ and $z_{1_1} = IV_1$.
then return C^f;

Figure 5.8: Game G12 as final game, and Adversary $\mathcal{B}$ using G12 as Adversary $\mathcal{A}$.

5.4 Conclusion

We presented a new variant, SpPad, of SpAEP using Sponge construction in ideal permutation model. A different but practical security notion over trapdoor one-way functions enables the use of SpPad with trapdoor one-way functions like El-Gamal. In addition to streaming at encryption side, it also provides streaming at decryption side by removing the dependency of randomness recovery from entire ciphertext/plaintext. Overall, with the combination of versatile Sponge structure and OW-PCA security assumption, SpPad–Pe achieves lower ciphertext overhead, streaming at encryption and decryption, lower computation cost compared to previous similar works.

5.4.1 Subsequent scope

With the help of Sponge structure, we have achieved lower ciphertext overhead, "on-the-fly" encryption and decryption, stronger security (IND-CCA) from weakly one-way secure asymmetric cryptosystem (both deterministic and probabilistic), support for long messages, and better computation efficiency. Security proof is based on ideal permutation model which is different from regular RO model in practice, provide similar security with less heuristic approach. We achieved these results in step by step manner by instantiating the "General view of OAEP+" of section 3.1.3 with Sponge structure. The *general view* we adopt is generic in nature and opens up more options to build different padding schemes for a CCA-secure asymmetric encryption scheme from one-way cryptosystems. A question arises about describing security of *general view*, with appropriate modifications to achieve maximum properties like we obtained with specific Sponge structure. This secure and efficient *general view* would be a competitive alternate option when compared to existing generic constructions like FO-transform [55], REACT [83] and GEM [40]. With the aim of having such generic framework, we make an attempt to provide an answer to this question in next chapter.

Viewing versatility of Sponge structure and usage of OAEP type padding in signcryption scheme opens up another scope of work to apply and use Sponge structure for efficient signcryption schemes. With this aim of signcryption schemes with Sponge structure, we carry on to next chapter.

Chapter 6

Real time CCA-secure Encryption for Arbitrary Long messages

Contents

In this chapter, we introduce a generic framework for building CCA-secure encryption for arbitrary long messages using a symmetric encryption scheme, a weakly one-way secure asymmetric cryptosystem, and hash functions.

First, we explain some existing works that propose a generic framework for CCA-secure encryption for arbitrary long messages using a symmetric encryption scheme, a one-way secure cryptosystem, and hash functions. We elaborate some limitations which are common in those works along with a comparison table to provide targeted motivation. Next, we explain features and comparison of our proposal REAL compared to existing schemes as part of our contribution. Following a detailed description of a generic framework (REAL), we describe two different versions of this framework (REAL-1, REAL-2) suitable to different system requirements. We also provide security proofs of both versions. At the end of this chapter, we conclude the chapter and discuss the scope of subsequent work.

6.1 Background

From previous chapters we know, public-key encryption of arbitrarily long messages is a very important issue. A hybrid approach that uses a combination of PKE and symmetric encryption is a common solution. Shoup presented a generic construction of hybrid encryption called the key/data encapsulation mechanism (or KEM/DEM) [97]. In [1], Abe, Gennaro, and Kurosawa proposed a modification that is called the Tag-KEM/DEM framework. Key-encapsulation mechanism (KEM) can be implemented using either OAEP-type encryption or CCA-secure PKE. Data encapsulation mechanism (DEM), on the other hand, is based on a symmetric encryption algorithm (such as AES) to process long messages. Chow et al. [37] obtained a generic and efficient transformation targeting embedded devices. They proposed an identity based encryption from any identity based KEM scheme that is secure against chosen-ciphertext attack (CCA). Another scheme called OAEP++ (see [27,28,63]), which describes IND-CCA secure PKEs, claimed to be computationally efficient but constructed only from any deterministic OW asymmetric primitives.

Fujisaki and Okamoto [54] proposed a generic framework. They showed how to convert any OW-secure PKE into a CCA-secure PKE in the random oracle model. Pointcheval [90] obtained a similar result but using any partially trapdoor one-way function in the random oracle model. Improved versions of the Fujisaki-Okamoto (FO) scheme were proposed in [40,83]. These constructions reduce the ciphertext overhead and remove the need for re-encryption during decryption. Security of these construction is proved under the the OW-PCA assumption for underlying asymmetric encryption, which is a slightly stronger notion than OW used in the FO scheme.

6.1.1 Limitation of previous works

Table 6.1 summarizes constructions of CCA-secure public key encryption schemes. In particular, OAEP-type schemes are built from deterministic trapdoor one-way permutations only. Hybrid encryption schemes suffer from a high ciphertext overhead except for RKEM-DEM [25]. However, RKEM-DEM requires stronger security assumptions. Boyen [31] described an efficient PKE with a minimum ciphertext overhead. Moreover, the security is based on the DH assumption and RO.

The FO transform [55] improved these works and then it is successfully enhanced by [40, 55, 83]. These works provide different generic constructions, where Pe can be either a trapdoor one-way permutation or a trapdoor one-way function. Besides, the security requirements for Pe are weakened. In this work, we focus on the works [40, 55, 83], which are proven in the RO model and offer best efficiency as well as generic construction.

Fujisaki and Okamoto [55] formulated their framework (FO transform) by using hybrid encryption efficiently. It deploys three primitives: a OW secure Pe, an IND-CPA secure symmetric encryption, and two random oracles. Although the design is quite efficient and requires weak security assumptions, the FO transform also suffers from a high-ciphertext overhead equal to the output length of asymmetric part (Pe). A similar high ciphertext overhead is a common weakness of hybrid encryption schemes and also of generic transformations described in [54, 55, 90]. The FO transform is also inherently sequential, i.e. asymmetric-key encryption follows symmetric key encryption. More precisely, in the FO transform complete cryptogram stream obtained from symmetric-key encryption has to be hashed before asymmetric-key encryption. The FO transform incurs higher computation time, especially for very long messages. This higher computation time and sequential nature prevents the FO transform being used for streaming (on-the-fly) encryption, where the length of message stream may not be known in advance. Another limitation of this work is the need of re-encryption during decryption. The re-encryption is found to be a necessary evil because of lower security requirements. Other than the extra computation load of re-encryption, there is an additional delay in the overall decryption process as the re-encryption process is executed in the end, after passing over the complete ciphertext. If the re-encryption process can be done in parallel with other operations during decryption, then it could reduce the computation overhead.

	Schemes	Asymmetric primitive (Pe)		Symmetric primitives		Ciphertext overhead	Long message support
		Security Requirement	Type	Security Requirement	Type		
OAEP Type schemes	OAEP [13]	POW	Trap.Perm. (e.g., RSA)	RO	2 Hash	$3k$	✗
	OAEP+ [98]	OW	Trap.Perm.	RO	3 Hash	$3k$	✗
	OAEP-3R [87]	OW	Trap.Perm.	RO	3 Hash	$2k$	✗
	OAEP-4X [2]	OW	Trap.Perm.	RO + IND-CPA	5 Hash + 1 SE	k	✓
	OAEP++ [28]	OW	Trap.Perm.	RO + IND-CPA	3 Hash + 1 SE	$2k$	✓
	SpAEP [8]	OW	Trap.Perm.	Ideal permutation (P)	P based 1 Hash + 1 SE	$2k$	✓
Hybrid Encryption	Cramer-Shoup [42]	DDH	DDH based (e.g.,ElGamal)	(KDF+TCR) + IND-CCA	2 Hash + 1 SE	$\ell_{pe} + co_{pe} + co_{\mathsf{S}}$	✓
	Cramer-Shoup [43]	IND-CCA KEM	Any Pe	IND-CCA	SE	$\ell_{pe} + co^*_{pe} + co_{\mathsf{S}}$	
	Kurosawa-Demstad KEM-DEM [68]	DDH	DDH based	(TCR+KDF + MAC),+ IND-CCA	3 Hash +1 SE	$\ell_{pe} + co_{pe} + co_{\mathsf{S}}$	✓
	RKEM-DEM [25]	IND-CCA+ RoR-CCA	Any Pe	IND-CCA	1 SE	$k + co_{pe} + co_{\mathsf{S}}$	✓
	Miniature CCA-PKE [31]	DDH	DDH based	RO	3 Hash	co_{pe}	✗
Generic Construction	FO- Transformation [55]	OW	Any Pe	RO + IND-CPA	2 Hash + 1 SE	$\ell_{pe} + co_{pe}$	✓
	REAL (**Our Result**)	OW	Any Pe	RO + IND-CPA	3 Hash + 1 SE	$2k + co_{pe}$	✓

Table 6.1: Comparison among some techniques results in IND-CCA secure schemes: We compare our scheme REAL against OAEP-type schemes, hybrid encryptions schemes and FO transform [55].Some abbreviations used in table are :
Random oracle model(RO), Partial One-wayness(POW), One-wayness(OW), chosen plaintext attack indistinguishability(IND-CPA), chosen ciphertext attack indistinguishability(IND-CCA), Key derivative function(KDF), Target collision resistant(TCR), message authentication code(MAC) respectively.
"Trap.Perm." refers to underlying deterministic one-way asymmetric cryptosystem, whereas "Any Pe" includes both deterministic and probabilistic one-way asymmetric cryptosystem(Pe).
k is security parameter, $\ell_{pe}(\geq k)$ is input length of asymmetric primitive , co_{Pe} is ciphertext overhead of asymmetric primitive and co_{S} is ciphertext overhead of symmetric encryption (SE) primitive. We consider IND-CPA SE is length preserving while providing value of ciphertext overhead.
co^*_{pe} might be 0 or co_{pe} depending upon KEM.

In [83], Okamoto et al. proposed scheme, named REACT, which overcame the limitation of re-encryption during decryption needed in FO transform [55]. This improvement of performance is achieved at the cost of a stronger security assumption, OW-PCA, on the asymmetric primitive Pe. This security assumption is easily satisfied by ElGamal [57] encryption under GDH assumption [82]. However, improvement in performance also increase the length of cryptogram. Moreover, while hashing, system needs to store either complete message or ciphertext. The hash function also needs to process both input (message) and output (ciphertext) of symmetric primitive. This results in a high memory demand for long message and the hash computation is equivalent to double pass.

In [40], Coron et al. focus on reducing the ciphertext overhead of the scheme REACT from [83]. Their scheme GEM has ciphertext overhead equivalent to the overhead of the FO transform, which is smaller than REACT [83]. The reduction in ciphertext overhead achieved at cost of losing the "on the fly" encryption/decryption option.

In the work [45], Cui et al. propose a generic CCA-secure scheme, named ROC, inspired from REACT [83] and GEM [40]. The ROC scheme incurs a lower ciphertext expansion compared with the scheme from [40, 55, 83]. On the downside, ROC can process fixed length messages only and does not support on-the-fly encryption/decryption.

Table 6.2 provides a summary of the works discussed above. In short, the use of re-encryption and weak security assumptions are found to be inversely proportional to each other; and ciphertext overhead, streaming option and memory requirements vary because of using different internal functions.

schemes	Asymmetric Encryption	Re-Encryption	Message length	Ciphertext overhead	# of other functions	on the fly Enc	on the fly Dec
FO [55]	OW-CPA	Yes	Unrestricted	$\ell + co_{pe}$	2Hash + 1 SE	No	Yes
REACT [83]	OW-PCA	No	Unrestricted	$\ell + co_{pe}$ + k	2Hash + 1 SE	Yes	Yes
GEM [40]	OW-PCA	No	Unrestricted	$\ell + co_{pe}$	3Hash +1SE	No	Yes
ROC [45]	OW-PCA	No	Restricted	$co_{\mathsf{Pe}} + k$	2 Hash	No	No

Table 6.2: Generic CCA transformations:
"$\ell + co_{pe}$" is output length and ℓ is input length of asymmetric encryption
"k" is security parameter, "SE" is Symmetric encryption
"OW" is one-wayness, "CPA" is chosen plaintext attack and "PCA"" is plaintext checking attack

6.1.2 Motivation

Our primary motivation of this work is to achieve a "real-time" encryption/decryption property with a lower ciphertext overhead, which is either missing in previous works or requires more memory. Real-Time algorithms process data streams, in which the input is presented as a sequence of items and can be examined in just one pass. These algorithms have limited memory available to them (much less than the input size) and also limited processing time. These algorithms have many applications, especially for processing long streams of data (movies online, music etc.), where their lengths may not be known in advance.

Data streaming is one of the drivers for calling the CAESAR [18] competition. However, the CAESAR call relates to symmetric key cryptography. The asymmetric key cryptography, which has been designed to process relatively short messages, somehow has been overlooked. Hybrid cryptography combines asymmetric with symmetric cryptography and allows to process very long messages.

To maintain a focus on security, many works (discussed already) have been proposed to build IND-CCA secure schemes for working with long messages under different security models. However, the "real-time" encryption and decryption remains an open problem, which is being addressed in the work.

Sponge instantiated version of a "Generic view of OAEP+" as described in Section 3.1.3 has shown good results to achieve our aims. We propose a modified version of this "Generic view of OAEP+" to achieve better result compared to existing generic schemes.

6.1.3 One-time Symmetric Encryption

A one-time symmetric encryption scheme $\mathsf{S} = (\mathcal{K}, S.Gen, S.Enc, S.Dec)$ consist of four algorithms defined as follows:

1. One time key value generation: Scheme S requires a random secret string K uniformly drawn from space $\{0,1\}^k$. We denote this as $K \xleftarrow{\$} \{0,1\}^k$ or alternatively $K \leftarrow \mathcal{K}(\cdot)$. The value K acts as key but K is freshly re-sampled from its space upon each execution of S.Enc.

2. Long term key value generation: $S.Gen$ defines a secret key Key from secret key space, $Key \leftarrow S.Gen(\cdot)$. In cases where no long term key Key is required then secret key space is $\emptyset$.

3. Encryption: The encryption algorithm $S.Enc$ takes as input a message M from the message space $\mathbb{M}$ and outputs a ciphertext C from the ciphertext space $\mathbb{C}$. More precisely, $S.Enc_{K,Key} : \mathbb{P} \rightarrow \mathbb{C}$

Correctness condition for S is as follows: If $K \leftarrow \mathcal{K}(\cdot)$, $Key \leftarrow S.Gen(\cdot)$ and $C \leftarrow S.Enc_{K,Key}(M)$ for any $M \in \mathbb{M}$ then $S.Dec_{K,Key}(C) = M$. This condition guarantees that decryption must give the same correct message M, when a ciphertext C is decrypted using the same (K, Key) as has been used for encryption. Indistinguishability of encryptions (IND) for one-time encryption also called find-guess security, is defined by the following game.

Game $\mathsf{IND\text{-}OT}()$

1. $d \leftarrow \{0, 1\}$
2. $(m_0, m_1, s) \leftarrow \mathcal{B}_1(1^k)$
3. $K \leftarrow \mathcal{K}(\cdot)$; $c^* = S.Enc_K(m_d)$
4. $d' \leftarrow \mathcal{B}_2(s, c^*)$

S is $\mathsf{IND\text{-}OT}$ if and only if for any couple of PPT algorithm $\mathcal{B}^{ot}_{\mathsf{S}} = (\mathcal{B}_1, \mathcal{B}_2)$,

$$Adv^{ot}_{\mathcal{B},\mathsf{S}} = |2\Pr[d' = d] - 1| = |\Pr[d' = d] - \Pr[d' \neq d]| \in negl(k)$$

The m_0 and m_1 generated by $\mathcal{B}_1$ should be in $\mathbb{M}$.

Another similar notion, defined as "indistinguishability of ciphertext" or "indistinguishability from random bits", is as follows: Game $\mathsf{IND\$\text{-}OT}()$

1. $d \leftarrow \{0, 1\}$
2. $(m, s) \leftarrow \mathcal{B}_1(1^k)$
3. $K \leftarrow \mathcal{K}(\cdot)$; $c_0 = S.Enc_K(m)$
4. $c_1 \leftarrow \{0, 1\}^{|c_0|}$
5. $d' \leftarrow \mathcal{B}_2(s, c_d)$

S is $\mathsf{IND\$\text{-}OT}$ if and only if for any couple of PPT algorithm $\mathcal{B}^{\$ot}_{\mathsf{S}} = (\mathcal{B}_1, \mathcal{B}_2)$,

$$Adv^{\$ot}_{\mathcal{B},\mathsf{S}} = |2\Pr[d' = d] - 1| = |\Pr[d' = d] - \Pr[d' \neq d]| \in negl(k)$$

The m generated by $\mathcal{B}_1$ should be in $\mathbb{M}$.

We assume $|S.Enc(K, M)| = \mathsf{Clen}(|M|)$ for some linear-time computable "ciphertext length function" Clen. The scheme S is said to be length preserving if $\mathsf{Clen}(|M|) = |M|$. We require S to be secure against one-time attacks. An adversary $\mathcal{B}$ has to distinguish the output of $S.Enc(K, M)$ from a randomly chosen bit-string of length $\mathsf{Clen}(|M|)$, where K is randomly chosen and the message M is chosen by $\mathcal{B}$.

We agree that this notion of "indistinguishability from random bits(IND$)" is stronger than traditional IND, but in practice IND$ seems more practical and typical encryption schemes seem to achieve IND$ if they achieve IND. This argument is supported and well discussed by Rogaway in [95]. "It is easy to verify that the ind$-notion of security implies the ind-notion, and by a tight reduction, while ind does not imply ind$ at all. Furthermore, it usually seems to be no extra trouble–indeed often it is slightly simpler–to directly demonstrate that some scheme achieves ind$-security.[...] Finally, we find ind$ seems to us conceptually simpler and easier to work with." [95]. Moreover, in practice, encryption schemes are supposed to give randomized output thats why nonce based cryptography is introduced.

Although previous works have provided that their hybrid PKE schemes using a symmetric encryption have IND-CPA security, they have suggested using a one-time pad scheme or a pseudorandom generator over one-time session key to use as a symmetric encryption scheme which is similar to using IND$-CPA.

6.2 Contribution

This work proposes a generic framework that converts any OW asymmetric primitive (Pe) into a CCA-secure and efficient PKE. Apart from Pe, the framework requires an one-time symmetric encryption (S) and three hash functions namely generator G, hider H and final F. All hash functions are considered as random oracles. REAL denotes encryption scheme constructed using our framework. Our contribution overcomes the limitations, mentioned in Section 6.1.1, of previous works [40, 55, 83].

Security of REAL is proven in the random oracle model. We have provided two version namely REAL-1 and REAL-2. REAL-1 is with re-encryption mechanism like the FO transform with OW security assumption on asymmetric primitive Pe.

For $\mathtt{REAL}$-2, we assume that Pe is $\mathsf{OW\text{-}PCA}$ secure in order to avoid re-encryption. Security assumptions and the security of the scheme are quite similar to previously published schemes [40, 55, 83]. However, our scheme handles streams of message with a very low ciphertext overhead. Our scheme targets real-time encryption that is applied in memory constrained devices and streaming applications in networks. $\mathtt{REAL}$-2 version should be used in case of Pe as trapdoor one-way permutations, where no re-encryption mechanism is needed and $\mathsf{OW\text{-}PCA}$ assumption is same as OW.

$\mathtt{REAL}$ (both $\mathtt{REAL}$-1 and $\mathtt{REAL}$-2) achieves a lower ciphertext overhead compared with the schemes from [40, 55, 83]. Let us denote co_S and co_Pe to be ciphertext overheads of S and Pe, respectively. Let k be the length of random strings used in $\mathtt{REAL}$. Then the total ciphertext overhead of $\mathtt{REAL}$ will be $\mathsf{co}_\mathsf{S} + \mathsf{co}_\mathsf{Pe} + 2k$. Due to hybrid nature of the schemes FO transform [55], REACT [83] and GEM [40], the ciphertext overhead of FO transform and GEM is $(\mathsf{co}_\mathsf{S} + \mathsf{co}_\mathsf{Pe} + \ell_{pe})$. For the scheme REACT, the overhead is $(\mathsf{co}_\mathsf{S} + \mathsf{co}_\mathsf{Pe} + \ell_{pe} + k)$.

During encryption, the computation time of $\mathtt{REAL}$ is lower than the FO transform [55] and GEM [40]. In the FO transform, asymmetric-key encryption has to wait until the symmetric-key encryption is completed. In $\mathtt{REAL}$ asymmetric primitive operation can start just after the initial partial output of symmetric cipher operation. Let $t_{asym}^{\ell_{pe}}$ or simply t_{asym} be computation time of asymmetric primitive Pe for its fixed input length ℓ_{pe}. Let t_{sym}^{n} be computation time of symmetric cipher for input length n, and $t_{sym}^{\ell_{pe}} (< t_{sym}^{n})$ be the computation time of symmetric cipher to output ℓ_{pe} bits, where $\ell_{pe} < n$. The resulting computation time of FO transform and GEM [40] will be $(t_{asym} + t_{sym}^{n})$. On the other hand, $\mathtt{REAL}$ will have $\mathtt{max}$[1]$(t_{asym} + t_{sym}^{\ell_{pe}},\ t_{sym}^{n})$ computation time (time resulting from computing hash functions is ignored). This decrease in the computation time in $\mathtt{REAL}$ compared to [40,55] might appear to be small because in general $t_{asym} > t_{sym}^{n}$. However, it is significant in case of very long messages when n is sufficiently large than ℓ_{pe} which results in $t_{sym}^{n} > t_{asym}$.

If we would like to use weak assumptions on asymmetric primitive Pe, then $\mathtt{REAL}$-1 inherits re-encryption mechanism during decryption like the FO transform. Let t_{asymd} be computation time of asymmetric primitive during decryption. In FO, the re-encryption is done at the end of decryption of the complete ciphertext and cannot be computed in parallel with other operations. The resulting decryption

[1] $\mathtt{max}$ function returns a maximal value amongst the parameters.

computation time of FO transform will be $t_{asymd} + (t_{asym} + t^n_{sym})$. In REAL-1, re-encryption process can be done as soon as the asymmetric decryption is completed. This enables us to perform re-encryption process in parallel with other operations including symmetric-key encryption. In case of REAL-1 decryption time will be $t_{asymd} + \texttt{max}(t_{asym}, t^n_{sym})$, which is lower than for the FO transform. A lower ciphertext overhead, lower encryption and decryption computation time and on-the fly encryption/decryption makes REAL-1 a better candidate.

If we compare REAL-2 to the schemes GEM [83] and REACT [40], then they have a similar decryption time and there is no need for re-encryption. REAL also provides a streaming capability, which can be a very useful feature, specially for broadcast systems. This feature applies because once the asymmetric part is processed (encrypted/decrypted), data can be streamed using the symmetric encryption/decryption. Although REACT [83] also provides streaming during encryption as well as decryption, while GEM [40] provides during decryption, the ciphertext overhead of REAL is lower than both [83] and [40]. If Pe is deterministic one-way asymmetric cryptosystem with weak security assumption then REAL-2 should be chosen. OAEP++ from [28] has similar features compared to REAL-2, but REAL-2 also provides streaming option during decryption and an overall generic structure.

Finally, it is interesting to observe that the framework we use is quite different from a regular hybrid encryption. Recall that the hybrid encryption uses two systems, namely KEM and DEM with a clear delineation between the two. In our framework, the two overlap. Table 6.3 overviews most prominent CCA-secure generic PKEs. The last line in the table shows our REAL construction. Note that it compares favorably with other constructions.

schemes	Asymmetric Encryption	Re-Encryption	Message length	Ciphertext overhead	# of other functions	on the fly Enc	on the fly Dec
FO [55]	OW-CPA	Yes	Unrestricted	$\ell + co_{pe}$	2Hash + 1 SE	No	Yes
REACT [83]	OW-PCA	No	Unrestricted	$\ell + co_{pe}$ + k	2Hash + 1 SE	Yes	Yes
GEM [40]	OW-PCA	No	Unrestricted	$\ell + co_{pe}$	3Hash +1SE	No	Yes
(Our Result) REAL	OW-CPA	Yes	Unrestricted	$co_{\mathsf{Pe}} + 2k$	3Hash + 1 SE	Yes	Yes
	OW-PCA	No					

Table 6.3: Generic CCA transformations:
"$\ell + co_{pe}$" is output length and ℓ is input length of asymmetric encryption
"k" is security parameter, "SE" is Symmetric encryption
"OW" is one-wayness, "CPA" is chosen plaintext attack and "PCA"" is plaintext checking attack

6.3 Real time CCA-secure Encryption for Arbitrary Long messages (REAL)

In this section, we introduce our Real time CCA-secure Encryption for Arbitrary Long messages (REAL). REAL can be used as an implementation template. A graphical representation of REAL is shown in Fig. 6.1.

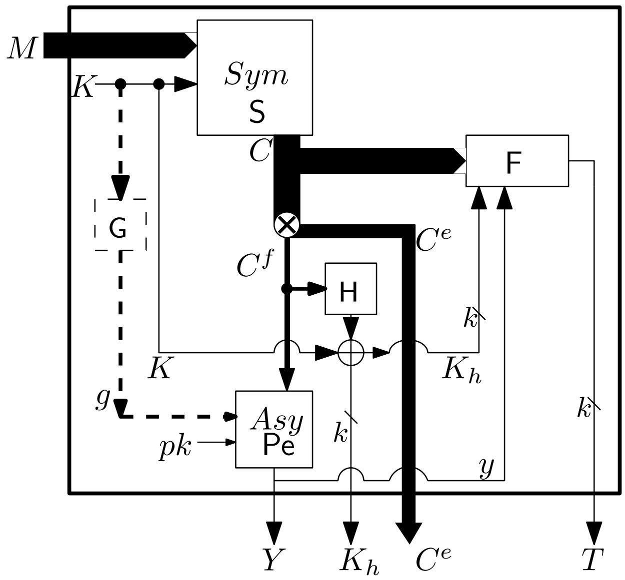


Figure 6.1: Public key scheme REAL is constructed using One-time symmetric encryption scheme (S) and Hash functions G, H and F, and an OW-CPA secure asymmetric primitive Pe. S takes the arbitrary long message M and a randomly generated K as input and then outputs C. C gets split into C^f and C^e as $C = C^f||C^e$ with $|C^f| = \ell - k$, where ℓ is input size of Pe. Hash function H takes C^f as input and its output is xored with K to produce K_h. Hash function G takes input K and outputs random coin g. $\mathsf{Enc}_{\mathsf{pk}}$ of Pe takes C^f as input with g as random coins *if needed*, and outputs Y. Final hash function F takes $(C^f||K_h||C^e||Y)$ as input and outputs T. Final output of REAL is $Y||K_h||C^e||T$.
Dashed line in figure represents optional g which is not required in case of deterministic Pe or in case Pe is considered as OW-PCA secure.

6.3.1 Generic Construction with Pe as OW : REAL-1

The building blocks for REAL-1 are:

1. an OW-CPA asymmetric encryption scheme Pe: $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ of minimum input message size ℓ as described in Section 2.1,

2. one-time symmetric encryption scheme $S : (S.Enc, S.Dec)$ for $\mathsf{Clen}(\cdot) \geq \ell$ as described in 6.1.3 and

3. Hash functions Generator $\mathsf{G} : \{0,1\}^k \to \texttt{COINS}$, Hider $\mathsf{H} : \{0,1\}^\ell \to \{0,1\}^k$ and Final $\mathsf{F} : \{0,1\}^* \to \{0,1\}^k$.(Modeled as RO)

A REAL-1 scheme is defined as a triplet of the following probabilistic polynomial-time (PPT) algorithms: $\langle GPKE.Gen, GPKE.Enc, GPKE.Dec\rangle$.

- $GPKE.Gen$ produces a private/public key pair$(\mathsf{pk}, \mathsf{sk})$ using $\mathsf{Gen}(1^k)$.
- $GPKE.Enc$ encrypts a message M of an arbitrary length and produces a cryptogram. Encryption proceeds according to the following steps:
  1. Take a message M and generate a random string $K \xleftarrow{\$} \{0,1\}^k$.
  2. $C = S.Enc(K, M)$,
  3. Split the C into C^f and C^e e.g., $C = C^f||C^e$, where $|C^f| = \ell$.
  4. $K_h = \mathsf{H}(C^f) \oplus K$, $g = \mathsf{G}(K)$
  5. $Y = \mathsf{Enc}_{\mathsf{pk}}(C^f; g)$
  6. $T = \mathsf{F}(C^f||Y||K_h||C^e)$
  7. Output Final ciphertext $\chi = (Y||K_h||C^e||T)$.
- $GPKE.Dec$ recovers a message M from a ciphertext χ and is implemented as follows.
  1. Parse the ciphertext χ to extract its parts $\chi = (Y||K_h||C^e||T)$.
  2. $C^f = \mathsf{Dec}_{\mathsf{sk}}(Y)$,
  3. $K = \mathsf{H}(C^f) \oplus K_h$; $g = \mathsf{G}(K)$
  4. $M = S.Dec(K, C^f||C^e)$
  5. $T' = \mathsf{F}(C^f||Y||K_h||C^e)$
  6. $Y' = \mathsf{Enc}_{\mathsf{pk}}(C^f; g)$
  7. **If** $(T == T' \& Y == Y')$ **then** Return M **else** Return $\perp$.

Now we are ready to present a security proof of CCA security of REAL. We assume that H, G and F are independent Random oracles. As described in Section 2.2, the experiment of adversary $\mathcal{A}$ for REAL is as follows:

Experiment: $Exp^{ind-cca2}_{\texttt{REAL},\mathcal{A}}(k)$

---

1. $(\mathsf{pk}, \mathsf{sk}) \xleftarrow{\$} \mathsf{Gen}(1^k)$;
2. $(M_0, M_1, s) \leftarrow \mathcal{A}_1^{\mathsf{H}(\cdot),\mathsf{G}(\cdot),\mathsf{F}(\cdot),GPKE.Dec(\cdot)}$;
3. $d \xleftarrow{\$} \{0,1\}$
4. $\chi^* \leftarrow GPKE.Enc(M_d)$; ... one time encryption query
5. $d' \leftarrow \mathcal{A}_2^{\mathsf{H}(\cdot),\mathsf{G}(\cdot),\mathsf{F}(\cdot),GPKE.Dec(\cdot)}(\chi^*, s)$;
6. return d';

Theorem 6. *Given a* OW-CPA *asymmetric encryption primitive* Pe:(Gen, Enc, Dec), *a one-time secure encryption scheme* $\mathsf{S} = (S.Enc, S.Dec)$ *and random oracles* H, G *and* F, *then the construction of* REAL *defined in Section 6.3.1 is* IND-CCA *secure. The success probability of any adversary* $\mathcal{A}$ *is*

$$Pr[Exp^{ind-cca2}_{\texttt{REAL},\mathcal{A}}(k) = d] \leq \frac{1}{2} + Adv^{\$ot}_{\mathcal{B},S} + Succ^{OW}_{\mathcal{C},\mathsf{Pe}} + \frac{q_d + q_g}{2^k} + \frac{q_d}{2^\lambda},$$

where q_d *is number of queries to the decryption oracle and* q_g *is number of queries to the* G *oracle.* $\mathcal{B}$ *is an adversary which tries to break one-time security of* S *with an advantage of* $Adv^{ot}_{\mathcal{B},S^{P_1}}$. $\mathcal{C}$ *is an adversary that finds the complete input* X *of* Pe *given* Y *such that* $Y = \mathsf{Enc}_{\mathsf{pk}}(X; g)$, *for some randomness* g *if present, without knowing* sk. $Succ^{OW}_{\mathcal{C},\mathsf{Pe}}$ *is an success advantage that a particular adversary* $\mathcal{C}$ *has in breaking* OW-CPA *security of* Pe.

Proof. Each game uses the following oracles:

- *GPKE.Enc* and *GPKE.Dec* perform *encryption* and *decryption*, respectively,
- Random oracles $\mathsf{F} : \{0,1\}^* \to \{0,1\}^k$, $\mathsf{H} : \{0,1\}^\ell \to \{0,1\}^k$ and $\mathsf{G} : \{0,1\}^k \to \texttt{COINS}$.
- *S.Enc* and *S.Dec* are internal function access to *GPKE.Enc* and *GPKE.Dec* respectively.

As *encryption*, *decryption*, H ,G and F are public oracles, they are accessible to the adversary $\mathcal{A}$, where $\mathsf{H}^{\mathcal{A}}$, $\mathsf{G}^{\mathcal{A}}$ and $\mathsf{F}^{\mathcal{A}}$ are interface through which $\mathcal{A}$ access H ,G and F oracles. In each game, the following lists are maintained: $I^{\mathcal{A}}_{\mathsf{H}}$ by $\mathsf{H}^{\mathcal{A}}$, $I^{\mathcal{A}}_{\mathsf{G}}$ by $\mathsf{G}^{\mathcal{A}}$, $I^{\mathcal{A}}_{\mathsf{F}}$ by $\mathsf{F}^{\mathcal{A}}$, I_{H} by H, I_{G} by G and I_{F} by F.

We will use the game technique [15,16]. We start from the original CCA game as defined in Section 2.2. $Exp_{\mathtt{REAL1},\mathcal{A}}$ Or $Exp^{ind-cca2}_{\mathtt{REAL-1},\mathcal{A}}(k) = d$ denote the event that $\mathcal{A}$ outputs $d' = d$, where $d \xleftarrow{\$} \{0,1\}$. We want to show that $Pr[Exp_{\mathtt{REAL1},\mathcal{A}}]| = \frac{1}{2} + negl(k)$. We slightly change REAL-1 into a sequence G0, G1, ..., G10 such that:

$$\begin{aligned} Pr[Exp_{\mathtt{REAL1},\mathcal{A}}] &= Pr[Exp_{G0,\mathcal{A}}] \\ Pr[Exp_{G(i-1),\mathcal{A}}] &= Pr[Exp_{Gi,\mathcal{A}}] + negl(k) \qquad \forall 1 \leq i \leq 9 \\ Pr[Exp_{G10,\mathcal{A}}] &= \frac{1}{2} \end{aligned}$$

In games G0 to G5, we make changes in *encryption* oracle along with some changes in H, F oracle to achieve that d of M_d is independent of all previous queries and their responses from *encryption* oracle. In games G6 to G10, we make small incremental changes in the *decryption* oracle and make it independent of sk. Challenge ciphertext χ^* has $C^*, Y^*, C^{f*}, C^{e*}, K^*, h^*, K^*_h, g^*, T^*$ as corresponding internal values during computation of challenge query.

Game G0: This game perfectly simulates the REAL-1.

$$Pr[Exp_{\mathtt{REAL1},\mathcal{A}}] = Pr[Exp_{G0,\mathcal{A}}].$$

The game G0 is the same as original CCA game of PKE.

Game G0: Initialize $I_{\mathsf{F}} = I_{\mathsf{H}} = I_{\mathsf{G}} = I^{\mathcal{A}}_{\mathsf{F}} = I^{\mathcal{A}}_{\mathsf{H}} = I^{\mathcal{A}}_{\mathsf{G}} = \emptyset$, $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^k)$.

On Encryption-Query(M_d)

1. $K^* \xleftarrow{\$} \{0,1\}^k$
2. $C^* = S.Enc(K^*, M_d)$
3. $C^{f*}||C^{e*} = C^*$
4. $K_h^* = \mathsf{H}(C^{f*}) \oplus K^*$
5. $g^* = \mathsf{G}(K^*)$
6. $Y^* = \mathsf{Enc}_{\mathsf{pk}}(C^{f*}; g^*)$
7. $T^* = \mathsf{F}(C^{f*}||Y^*||K_h^*||C^{e*})$
8. Return $\chi = (Y^*||K_h^*||C^{e*}||T^*)$

On Decryption-Query χ

1. $(Y||K_h||C^e||T) = \chi$
2. $(C^f) = \mathsf{Dec}_{\mathsf{sk}}(Y)$, $C = C^f||C^e$
3. $K = K_h \oplus \mathsf{H}(C^f)$; $g = \mathsf{G}(K)$
4. $Y' = \mathsf{Enc}_{\mathsf{pk}}(C^f; g)$
5. $T' = \mathsf{F}(C^f||Y||K_h||C^e)$
6. $M = S.Dec(K, C)$
7. **if** $T == T' \& Y == Y'$ **then**
   Return M
   else
   Return $\perp$

On F-Query $C^f||Y||K_h||C^e$

1. **if** $\exists\ T\ s.t.\ (C^f||Y||K_h||C^e, T) \in I_{\mathsf{F}}$ **then**
   return T
2. $T \xleftarrow{\$} \{0,1\}^k$
3. $I_{\mathsf{F}} = I_{\mathsf{F}} \bigcup \{(C^f||Y||K_h||C^e, T)\}$
4. return T;

On F$^{\mathcal{A}}$-Query $C^f||Y||K_h||C^e$

1. **if** $\exists\ T\ s.t.\ (C^f||Y||K_h||C^e, T) \in I^{\mathcal{A}}_{\mathsf{F}}$ **then**
   return T
2. $T = \mathsf{F}(C^f||Y||K_h||C^e)$
3. $I^{\mathcal{A}}_{\mathsf{F}} = I^{\mathcal{A}}_{\mathsf{F}} \bigcup \{(C^f||Y||K_h||C^e, T)\}$
4. return T;

On G-Query K

1. **if** $\exists\ g\ s.t.\ (K, g) \in I_{\mathsf{G}}$ **then**
   return g
2. $g \xleftarrow{\$} \{0,1\}^{Coins}$
3. $I_{\mathsf{G}} = I_{\mathsf{G}} \bigcup \{(K, g)\}$
4. return g;

On G$^{\mathcal{A}}$-Query K

1. **if** $\exists\ g\ s.t.\ (K, g) \in I^{\mathcal{A}}_{G}$ **then**
   return g
2. $g = \mathsf{G}(K)$
3. $I^{\mathcal{A}}_{\mathsf{G}} = I^{\mathcal{A}}_{\mathsf{G}} \bigcup \{(K, g)\}$
4. return g;

On H-Query C^f

1. **if** $\exists\ h\ s.t.\ (C^f, h) \in I_H$ **then**
   return h
2. $h \xleftarrow{\$} \{0,1\}^k$
3. $I_{\mathsf{H}} = I_{\mathsf{H}} \bigcup (C^f||h)$
4. return h;

On H$^{\mathcal{A}}$-Query C^f

1. **if** $\exists\ h\ s.t.\ (C^f, h) \in I^{\mathcal{A}}_H$ **then**
   return h
2. $h = \mathsf{H}(C^f)$
3. $I^{\mathcal{A}}_{\mathsf{H}} = I^{\mathcal{A}}_{\mathsf{H}} \bigcup \{(C^f, h)\}$
4. return v;

Figure 6.2: Game G0 of REAL-1

In Game G1: In decryption oracle, G1 adds a dummy event as $Flag_F \leftarrow new$, if the query to F oracle does not exists already in $I_{\mathsf{F}}^{\mathcal{A}}$, or $(C^f||Y||K_h||C^e, T) \notin I_{\mathsf{F}}^{\mathcal{A}}$. These changes are just conceptual. Therefore, $\Pr[Exp_{G1,\mathcal{A}}] = \Pr[Exp_{G0,\mathcal{A}}]$

Game G1 $\boxed{\textbf{G2}}$**:** Initialize $I_{\mathsf{F}} = I_{\mathsf{H}} = I_{\mathsf{G}} = I_{\mathsf{F}}^{\mathcal{A}} = I_{\mathsf{H}}^{\mathcal{A}} = I_{\mathsf{G}}^{\mathcal{A}} = \emptyset$, $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^k)$. $Flag_F \in \{old, new\}$

On Decryption-Query χ

$(Y||K_h||C^e||T) = \chi$; $Flag_F \leftarrow old$
$(C^f) = \mathsf{Dec}_{\mathsf{sk}}(Y)$, $C = C^f||C^e$
$K = K_h \oplus \mathsf{H}(C^f)$
$g = \mathsf{G}(K)$
$Y' = \mathsf{Enc}_{\mathsf{pk}}(C^f; g)$
If$\{C^f||Y||K_h||C^e, T\} \notin I_{\mathsf{F}}^{\mathcal{A}}$ **then** $Flag_F \leftarrow new$ $\boxed{\text{Return } \perp}$
$T' = \mathsf{F}(C^f||Y||K_h||C^e)$
$M = S.Dec(K, C)$
If$T == T' \& Y == Y'$ **then** Return M **else** Return $\perp$

Rest of oracles same as G0

Figure 6.3: Game G1 and G2: Compared to G0, G1 includes dummy lines of code shown as dashed line. G2 includes lines of code that are in solid line and dash line boxes.

Game G3 $\boxed{\textbf{G4}}$**:** Initialize $I_{\mathsf{F}} = I_{\mathsf{H}} = I_{\mathsf{G}} = I_{\mathsf{F}}^{\mathcal{A}} = I_{\mathsf{H}}^{\mathcal{A}} = I_{\mathsf{G}}^{\mathcal{A}} = \emptyset$, $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^k)$. $flag_{\mathsf{pk}} \in \{old, new\}$

On Decryption-Query χ

$(Y||K_h||C^e||T) = \chi$ $flag_{\mathsf{pk}} \leftarrow old$
$(C^f) = \mathsf{Dec}_{\mathsf{sk}}(Y)$, $C = C^f||C^e$
$K = K_h \oplus \mathsf{H}(C^f)$
$g = \mathsf{G}(K)$
If $\{C^f||Y||K_h||C^e, T\} \notin I_{\mathsf{F}}^{\mathcal{A}}$ **then** Return $\perp$
If $\{C^f, K_h \oplus K\} \notin I_{\mathsf{H}}^{\mathcal{A}}$ or $\{K, g\} \notin I_{\mathsf{G}}^{\mathcal{A}}$ **then** $flag_{\mathsf{pk}} \leftarrow new$ $\boxed{\text{Return } \perp}$
$Y' = \mathsf{Enc}_{\mathsf{pk}}(C^f; g)$
$M = S.Dec(K, C)$
If$Y == Y'$ **then** Return M **else** Return $\perp$

Rest of oracles same as G0

Figure 6.4: Game G3 and G4: G3 includes dummy lines of code shown as dashed line as compared to G2; G4 includes lines of code that are in solid line and dash line boxes.

Game G2: In Game G2, decryption oracle rejects the ciphertext if $Flag_F$ is *new*. Here difference between Game G1 and G2 arises when G2 rejects the ciphertext even when G1 has accepted them. This implies in G1, even on condition that $Flag_F$ is *new*, there is right matching of T and T'. It means that $\mathcal{A}$ knows an output value of the F function even without querying it. This could happen with probability of $\frac{q_d}{2^k}$, for q_d number of decryption queries. Therefore,

$|\Pr[Exp_{G2,\mathcal{A}}] - \Pr[Exp_{G1,\mathcal{A}}]| \leq \frac{q_d}{2^k}$

Game G2 and G3: Game G3 and G2 are the same except for a dummy event, which is added in decryption oracle. This event raises a $flag_{\mathsf{pk}}$ as *new* if query to H or G does not belong to $I^{\mathcal{A}}_{\mathsf{H}}$ or $I^{\mathcal{A}}_{\mathsf{G}}$, respectively. In decryption oracle, re-encryption validates Y as a right encryption of C^f under a particular g, where g is uniquely related to $\mathsf{G}(\mathsf{H}(C^f) \oplus K_h)$. The flag is just conceptual and does not bring any change of $\mathcal{A}$'s view.

$\Pr[Exp_{G3,\mathcal{A}}] = \Pr[Exp_{G2,\mathcal{A}}]$

Game G4: In Game G4, decryption oracle return $\perp$ if $flag_{\mathsf{pk}}$ is new. Here difference between Game G4 and G3 appears when G4 is rejecting the ciphertext even when G3 has accepted them. This implies in G3, even on condition that $flag_{\mathsf{pk}}$ is *new*, there is right matching of Y and Y'. It means $\mathcal{A}$ either has a right $K = h \oplus K_h$ for a random h value, where $(\{K, g\} \in I^{\mathcal{A}}_{\mathsf{G}} \wedge \{C^f, h\} \notin I^{\mathcal{A}}_{\mathsf{H}})$ or has a right g value randomly when $(\{K_h \oplus h, g\} \notin I^{\mathcal{A}}_{\mathsf{G}} \wedge \{C^f, h\} \in I^{\mathcal{A}}_{\mathsf{H}})$ so for $|g| = \lambda \geq k$

$|\Pr[Exp_{G4,\mathcal{A}}] - \Pr[Exp_{G3,\mathcal{A}}]| \leq (\frac{q_{\mathsf{G}}}{2^k} + \frac{q_d}{2^\lambda})$

Game G5: G5 and G4 are same except that decryption oracle runs without using sk. And decryption become independent from sk. $\Pr[Exp_{G3,\mathcal{A}}] = \Pr[Exp_{G2,\mathcal{A}}]$

Game G5: Initialize $I_{\mathsf{F}} = I_{\mathsf{H}} = I_{\mathsf{G}} = I^{\mathcal{A}}_{\mathsf{F}} = I^{\mathcal{A}}_{\mathsf{H}} = I^{\mathcal{A}}_{\mathsf{G}} = \emptyset$, $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^k)$.

On Decryption-Query χ	
1 $(Y \| K_h \| C^e \| T) = \chi$ 2 **if** $\exists T$ *in* $\{C^f \| Y \| K_h \| C^e, T\} \in I^{\mathcal{A}}_{F}$ *s.t* $\{C^f, K_h \oplus K\} \in I^{\mathcal{A}}_{H}$, $\{K, g\} \in I^{\mathcal{A}}_{G}$ *and* $Y = Enc_{pk}(C^f; g)$ **then** $\quad$ Return $M = S.Dec(K, C)$ **else** $\quad$ Return $\perp$	Rest of oracles same as G0

Figure 6.5: Game G5: Decryption oracle outputs same as G4, but independent from secret key sk.

Game $\boxed{\textbf{G6}}$ $\boxed{\textbf{G7}}$**:** Initialize $I_{\mathsf{F}} = I_{\mathsf{H}} = I_{\mathsf{G}} = I_{\mathsf{F}}^{\mathcal{A}} = I_{\mathsf{H}}^{\mathcal{A}} = I_{\mathsf{G}}^{\mathcal{A}} = \emptyset$, $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^k)$.

After Find Stage(AFS): $\boxed{K^* \xleftarrow{\$} \{0,1\}^k; \bar{K_h} \xleftarrow{\$} \{0,1\}^k; \bar{T} \xleftarrow{\$} \{0,1\}^k}$

On Encryption-Query(M_d)

1 $C^* = S.Enc(K^*, M_d)$
2 $C^{f*}||C^{e*} = C^*$
3 $K_h^* = \mathsf{H}(C^{f*}) \oplus K^*$ $\boxed{K_h^* = \bar{K_h}}$
4 $g^* = \mathsf{G}(K^*)$
5 $Y^* = \mathsf{Enc}_{\mathsf{pk}}(C^{f*}; g^*)$
6 $T^* = \mathsf{F}(C^{f*}||Y^*||K_h^*||C^{e*})$
$\boxed{T^* = \bar{T}}$
7 Return $\chi = (Y^*||K_h^*||C^{e*}||T^*)$

On $\mathsf{H}^{\mathcal{A}}$**-Query** C^f
(*Adversary's query)

1 $\boxed{\textbf{If } C^f == C^{f*} \textbf{ then}}$
$\boxed{Bad_f \leftarrow true}$ $\boxed{\text{Halt}}$
2 **if** $\exists\ h\ s.t.\ (C^f, h) \in I_H^{\mathcal{A}}$ **then**
∟ return h
3 $h = \mathsf{H}(C^f)$
4 $I_{\mathsf{H}}^{\mathcal{A}} = I_{\mathsf{H}}^{\mathcal{A}} \bigcup \{(C^f, h)\}$
5 return v;

On $\mathsf{F}^{\mathcal{A}}$**-Query** $C^f||Y||K_h||C^e$
(*Adversary's query)

1. $\boxed{\textbf{If } C^f == C^{f*} \textbf{ then}}$
$\boxed{Bad_f \leftarrow true}$ $\boxed{\text{Halt}}$
2. **if** $\exists\ T\ s.t.\ (C^f||Y||K_h||C^e, T) \in I_{\mathsf{F}}^{\mathcal{A}}$ **then**
∟ return T
3. $T = \mathsf{F}(C^f||Y||K_h||C^e)$
4. $I_{\mathsf{F}}^{\mathcal{A}} = I_{\mathsf{F}}^{\mathcal{A}} \bigcup \{(C^f||Y||K_h||C^e, T)\}$
5. return T;

Rest of oracles same as G5

Figure 6.6: Game G6 and G7: G6 includes dummy lines in dashed box compared to G5. G7 includes dummy lines and lines in solid line boxes.

Game G6: Game G6 and G5 are same, except a dummy event is added in $\mathsf{H}^{\mathcal{A}}$ and $\mathsf{F}^{\mathcal{A}}$ oracle in G6. This dummy event raises the event as $Bad_f \leftarrow true$ if Adversary queries C^{f*} to $\mathsf{H}^{\mathcal{A}}$ or $C^{f*}||*$ to $\mathsf{F}^{\mathcal{A}}$. K^* is chosen randomly before the encryption query but after "find" stage (AFS). A dummy variable $\bar{K_h}$ is also chosen at random. These changes are just conceptual ones. Therefore, $Pr[Exp_{G6,\mathcal{A}}] = Pr[Exp_{G5,\mathcal{A}}]$.

Game G7: In Game G7, We halt the game if $Bad_f \leftarrow true$ happens. In case this event does not happens then K_h^* and T^* in encryption oracle can be replaced by any random string like $\bar{K_h}$ and $\bar{T}$ respectively. This change is then essentially bounded by $(Pr[Bad_f] = Pr[(C^{f*}, *) \in I_{\mathsf{H}}^{\mathcal{A}} \vee (C^{f*}||*, *) \in I_{\mathsf{F}}^{\mathcal{A}}])$. Therefore, $|Pr[Exp_{G7,\mathcal{A}}] - Pr[Exp_{G6,\mathcal{A}}]| \leq Pr[Bad_f]$.
We continue games assuming halt has not happened. Later, we will calculate probability of Bad_f as a reduction to one-wayness of asymmetric encryption.

Game G8: In Game G8, some extra dummy random values $\bar{C} = \bar{C^f}||\bar{C^e} \in \{0,1\}^{Clen(M_d)}$ are chosen along with K^* and K_h^*. These changes are just dummy and conceptual therefore, $\Pr[Exp_{G8,\mathcal{A}}] = \Pr[Exp_{G7,\mathcal{A}}]$

Game G8 $\boxed{\textbf{G9}}$: Initialize $I_{\mathsf{F}} = I_{\mathsf{H}} = I_{\mathsf{G}} = I_{\mathsf{F}}^{\mathcal{A}} = I_{\mathsf{H}}^{\mathcal{A}} = I_{\mathsf{G}}^{\mathcal{A}} = \emptyset,, (\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^k)$.
(AFS): $K^* \xleftarrow{\$} \{0,1\}^k$; $K_h^* \xleftarrow{\$} \{0,1\}^k$; $T^* \xleftarrow{\$} \{0,1\}^k$;
$g^* = \mathsf{G}(K^*)$; $\bar{C} = \bar{C^f}||\bar{C^e} \xleftarrow{\$} \{0,1\}^{\mathsf{Clen}(M_d)}$

On Encryption-Query(M_d)	
1 $C^* = S.Enc(K^*, M_d)$ $\boxed{C^* = \bar{C}}$ 2 $C^{f*}\|\|C^{e*} = C^*$ 3 $Y^* = \mathsf{Enc}_{\mathsf{pk}}(C^{f*}; g^*)$ 4 Return $\chi = (Y^*\|\|K_h^*\|\|C^{e*}\|\|T^*)$	Rest of oracles same as G7

Figure 6.7: Game G8 and G9: G8 has some dummy change in code compared to G7 shown in dash box; this contains some dummy variables and replacement of original values from random values. G9 includes lines of dash box and solid line box.

Game G8 and G9: From G8 to G9, $C^* = S.Enc(K^*, M_d)$ is replaced by a random string $\bar{C}$. This change is bounded by adversary $\mathcal{B}$ attacking the (one-time) encryption scheme S . This transition from G8 and G9 can be understand from Game IND\$OT shown in figure 6.8. Therefore, $|Pr[Exp_{G9,\mathcal{A}}] - Pr[Exp_{G8,\mathcal{A}}]| \leq Adv_{\mathcal{B},S}^{\$ot}$

Game IND$OT

$b \leftarrow \{0,1\}$
$(m, \sigma) \leftarrow \mathcal{B}_1(1^k)$
$R \leftarrow \mathcal{R}(\cdot)$; $C_0 = S.Enc_R(m)$
$C_1 \leftarrow \{0,1\}^{|C_0|}$
$b' \leftarrow \mathcal{B}_2(\sigma, C_b)$

$\mathcal{B}_1(1^k)$

$(pk, sk) \leftarrow \mathsf{Gen}(1^k)$
$(M_0, M_1, s) \leftarrow \mathcal{A}_1^{\mathsf{H},\mathsf{G},\mathsf{F},GPKE.Dec}(pk)$
$d \leftarrow \{0,1\}$
$\sigma = (d, pk, s)$
Return (M_d, σ)

$\mathcal{B}_2(\sigma, C_b)$

$(d, pk, s) = \sigma$
$C^f || C^e = C_b$
$g \xleftarrow{\$} \mathtt{COINS}$; $R_h \xleftarrow{\$} \{0,1\}^k$; $T \xleftarrow{\$} \{0,1\}^k$
$y \leftarrow \mathsf{Enc}_{pk}(C^f; g)$
$\chi = y||R_h||C^e||T$
$d' \leftarrow \mathcal{A}_2^{\mathsf{H},\mathsf{G},\mathsf{F},GPKE.Dec_{sk,\chi}}(pk, s, \chi)$
if $d' = d$ **then**
  $b = 0$
else
  $b = 1$
Return b

$\mathcal{A}_1$ and $\mathcal{A}_2$ use same oracles H ,G ,F , $GPKE.Dec$ as in Game G8.

Figure 6.8: Game IND$OT: In-between Game G8 and G9 by Adversary $\mathcal{B}$.

Game G10: C^* is already a random string in G9 and independent from K^*, value of g^* can be chosen randomly independently from K^*. Now in G10, C^{f*} and C^{e*} are chosen randomly. All values for encryption oracle are computed randomly beforehand, therefore Y^* is computed on C^{f*} and g^*. These changes bring no change of the view of $\mathcal{A}$ and therefore are just conceptual. Therefore, $\Pr[Exp_{G10,\mathcal{A}}] = \Pr[Exp_{G9,\mathcal{A}}]$

In G10, encryption oracle runs independent from d bit of M_d. And finally game become independent from sk and bit d. $Pr[Exp_{G10,\mathcal{A}}] = \frac{1}{2}$

G10 is the final game as $\mathcal{A}$ for $\mathcal{C}$ who tries to find pre-image of a given random Y using pk. Probability of Bad_f is equivalent to the probability of breaking one-wayness of asymmetric primitive Pe , which gives $Pr[Bad_f] \leq Succ^{OW}_{\mathcal{C}_\mathcal{A},\mathsf{Pe}}$

This completes the proof.

Game G10: Initialize $I_{\mathsf{F}} = I_{\mathsf{H}} = I_{\mathsf{G}} = I_{\mathsf{F}}^{\mathcal{A}} = I_{\mathsf{H}}^{\mathcal{A}} = I_{\mathsf{G}}^{\mathcal{A}} = \emptyset$, $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^k)$.
(AFS): $g^* \xleftarrow{\$} \texttt{COINS}; K_h^* \xleftarrow{\$} \{0,1\}^k$; $T^* \xleftarrow{\$} \{0,1\}^k$ $C^{f*} \xleftarrow{\$} \{0,1\}^{\ell}; C^{*e} \xleftarrow{\$} \{0,1\}^{\mathsf{Clen}(M_d)-\ell}$; ; $Y^* = \mathsf{Enc}_{\mathsf{pk}}(C^{f*}; g^*)$

On Encryption-Query(M_d)

1 Return $\chi = (Y^*||K_h^*||C^{e*}||T^*)$

Rest of oracles same as G7

Figure 6.9: Game G10: G10 output is same as G9.

Adversary $\mathcal{C}$: Given $Y \xleftarrow{\$} \{0,1\}^{\ell + co_{pe}}$

Adversary $\mathcal{A}$: Initialize $I_{\mathsf{F}} = I_{\mathsf{H}} = I_{\mathsf{G}} = I_{\mathsf{F}}^{\mathcal{A}} = I_{\mathsf{H}}^{\mathcal{A}} = I_{\mathsf{G}}^{\mathcal{A}} = \emptyset$, $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^k)$.
(AFS): $g^* \xleftarrow{\$} \texttt{COINS}; K_h^* \xleftarrow{\$} \{0,1\}^k$; $C^{e*} \xleftarrow{\$} \{0,1\}^{\mathsf{Clen}(M_d)-\ell}$; $T^* \xleftarrow{\$} \{0,1\}^k$; $Y^* = Y$

Rest of Oracles same as G10

Finalization: If there exists a $(C^f, h) \in I_{\mathsf{H}}^{\mathcal{A}}$ or $(C^f||*, *) \in I_{\mathsf{F}}^{\mathcal{A}}$ such that $Y == \mathsf{Enc}_{\mathsf{pk}}(C^f; g)$ for any $(K, g) \in I_{\mathsf{G}}^{\mathcal{A}}$ then return C^f;

Figure 6.10: Adversary $\mathcal{C}$ using $\mathcal{A}$, where $\mathcal{A}$ uses oracles as simulated in G10

□

6.3.2 Generic Construction with Pe as OW-PCA : REAL-2

REAL-2 has similar encryption structure to REAL-1, except during decryption there is no need for re-encryption. As we have discussed already, re-encryption is a necessary evil in case of OW-CPA security assumption on Pe. If Pe is OW-PCA, then this re-encryption can be avoided. The building blocks for REAL-2 are:

1. an OW-PCA asymmetric encryption scheme Pe: $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ of minimum input message size (domain) ℓ as described in Section 2.1,
2. an one-time symmetric encryption scheme S for $\mathsf{Clen}(\cdot) \geq \ell$ as described in Section 6.1.3 and
3. hash functions $\mathsf{F} : \{0,1\}^* \rightarrow \{0,1\}^k$, $\mathsf{H} : \{0,1\}^{\ell} \rightarrow \{0,1\}^k$ (modeled as RO).

REAL-2 is defined as a triplet of the following probabilistic polynomial-time (PPT) algorithms: $\langle GPKE.Gen, GPKE.Enc, GPKE.Dec \rangle$.

- $GPKE.Gen$ produces a private/public key pair$(\mathsf{pk}, \mathsf{sk})$ using $\mathsf{Gen}(1^k)$.

- $GPKE.Enc$ encrypts a message M of an arbitrary length and produces a cryptogram. Encryption proceeds according to the following steps:

  1. Take a message M and generate a random string $K \xleftarrow{\$} \{0,1\}^k$.
  2. $C = S.Enc(K, M)$, ; $g \xleftarrow{\$} \texttt{COINS}$.
  3. Split C into C^f and C^e e.g., $C = C^f||C^e$, where $|C^f| = \ell$.
  4. $K_h = \mathsf{H}(C^f) \oplus K$
  5. $Y = \mathsf{Enc}_{\mathsf{pk}}(C^f; g)$
  6. $T = \mathsf{F}(C^f||Y||K_h||C^e)$
  7. Output final ciphertext $\chi = (Y||K_h||C^e||T)$.

- $GPKE.Dec$ recovers a message M from a ciphertext χ and is implemented as follows.

  1. Parse the ciphertext χ to extract its parts $\chi = (Y||K_h||C^e||T)$.
  2. $C^f = \mathsf{Dec}_{\mathsf{sk}}(Y)$, $C^f||C^e = C$
  3. $K = \mathsf{H}(C^f) \oplus K_h$
  4. $M = S.Dec(K, C)$
  5. $T' = \mathsf{F}(C^f||Y||K_h||C^e)$
  6. **If** $(T == T')$ **then** Return M **else** Return $\perp$.

A notable difference between `REAL`-2 and `REAL`-1 comes in the decryption procedure. In `REAL`-2, decryption is independent from random coins. In absence of random coins, plaintext-checking oracle ($\mathcal{O}^{PC}$) helps decryption simulator to verify right matching of candidate (input, output) pair of Pe. In this way re-encryption step of `REAL`-1 is replaced by $\mathcal{O}^{PC}$ during decryption simulator in `REAL`-2. One more difference is that `REAL`-2 has no need of G function.

Now we quickly present a security proof of CCA security of REAL-2. We assume that H and F are independent random oracles. As described in Section 2.2, the experiment of adversary $\mathcal{A}$ for $GPKE$ is as follows:

Experiment: $Exp^{ind-cca2}_{\mathtt{REAL}-2,\mathcal{A}}(k)$

---

1. $(\mathsf{pk}, \mathsf{sk}) \xleftarrow{\$} \mathsf{Gen}(1^k)$;
2. $(M_0, M_1, s) \leftarrow \mathcal{A}_1^{\mathsf{H}(\cdot),\mathsf{F}(\cdot),GPKE.Dec(\cdot)}$;
3. $d \xleftarrow{\$} \{0,1\}$
4. $\chi^* \leftarrow GPKE.Enc(M_d)$; ...one time encryption query
5. $d' \leftarrow \mathcal{A}_2^{\mathsf{H}(\cdot),\mathsf{F}(\cdot),GPKE.Dec(\cdot)}(\chi^*, s)$;
6. return d';

Theorem 7. *Given a* OW-PCA *asymmetric encryption primitive* Pe:(Gen, Enc, Dec)*, a one-time secure encryption scheme* S $= (S.Enc, S.Dec)$ *and random oracles* H *and* F*, then the construction of* REAL-2 *defined in Section 6.3.2 is* IND-CCA *secure. The success probability of an adversary $\mathcal{A}$ is*

$$Pr[Exp^{ind-cca2}_{REAL-2,\mathcal{A}}(k) = d] \leq \frac{1}{2} + Adv^{\$ot}_{\mathcal{B},S} + Succ^{OW-PCA}_{\mathcal{C},\mathsf{Pe}} + \frac{q_d}{2^k},$$

where q_d is number of queries to the decryption oracle. $\mathcal{B}$ is an adversary which tries to break one-time security of S *with an advantage of $Adv^{\$ot}_{\mathcal{B},S}$. $Succ^{\text{OW-PCA}}_{\mathcal{C},\mathsf{Pe}}$ is an success advantage that a particular adversary $\mathcal{C}$ has in breaking* OW-PCA *security of* Pe.

Proof. Each game uses the following oracles:

- *GPKE.Enc*, *GPKE.Dec* perform *encryption* and *decryption* respectively,
- Random oracles $\mathsf{F} : \{0,1\}^* \rightarrow \{0,1\}^k$, $\mathsf{H} : \{0,1\}^\ell \rightarrow \{0,1\}^k$.
- *S.Enc* and *S.Dec* are internal function access to *GPKE.Enc* and *GPKE.Dec* respectively.

As *encryption*, *decryption*, H and F are public oracles, they are accessible to the adversary, where $\mathsf{H}^{\mathcal{A}}$ and $\mathsf{F}^{\mathcal{A}}$ are interface through which adversary $\mathcal{A}$ access H , and F oracles. In each game, the following Lists are maintained: $I_{\mathsf{H}}^{\mathcal{A}}$ by $\mathsf{H}^{\mathcal{A}}$, $I_{\mathsf{F}}^{\mathcal{A}}$ by $\mathsf{F}^{\mathcal{A}}$, I_{H} by H and I_{F} by F

We will use game based proof technique [15, 16]. We start from the original CCA game as defined in Section 2.2. $Exp_{\mathtt{REAL2},\mathcal{A}}$ Or $Exp^{ind-cca2}_{\mathtt{REAL}-2,\mathcal{A}}(k) = d$ denote the event that $\mathcal{A}$ outputs $d' = d$ where $d \xleftarrow{\$} \{0, 1\}$. We want to show that $Pr[Exp_{\mathtt{REAL2},\mathcal{A}}]=\frac{1}{2} + negl(k)$. We slightly change $\mathtt{REAL}$-2 into a sequence G0, G1, …, G8 such that:
$Pr[Exp_{\mathtt{REAL2},\mathcal{A}}]= Pr[Exp_{G0,\mathcal{A}}]$
$Pr[Exp_{G(i-1),\mathcal{A}}]=Pr[Exp_{Gi,\mathcal{A}}]+negl(k)\ \forall 1 \leq i \leq 7$
$Pr[Exp_{G8,\mathcal{A}}]= \frac{1}{2}$

In games G0 to G3, we make small incremental changes in the *decryption* oracle and make it independent of sk.

In games G4 to G8, we make changes in *encryption* oracle along with some changes in H, F oracle to achieve that d of M_d is independent of all previous queries and their responses from *encryption* oracle.

Challenge ciphertext χ^* is having $C^*, Y^*, C^{f*}, C^{e*}, K^*, h^*, K_h^*, g^*, T^*$ as corresponding internal values during computation of challenge query.

Game G0: The game G0 is the same as original CCA game of PKE. This game perfectly simulates the $\mathtt{REAL}$-2.

$$Pr[Exp_{\mathtt{REAL2},\mathcal{A}}] = Pr[Exp_{G0,\mathcal{A}}].$$

Game G0: Initialize $I_{\mathsf{F}} = I_{\mathsf{H}} = I^{\mathcal{A}}_{\mathsf{F}} = I^{\mathcal{A}}_{\mathsf{H}} = \emptyset$, $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^k)$.

On Encryption-Query(M_d)

$K^* \xleftarrow{\$} \{0,1\}^k$;$g^* \xleftarrow{\$}$ COINS
$C^* = S.Enc(K^*, M_d)$
$C^{f*}||C^{e*} = C^*$
$Y^* = \mathsf{Enc}_{\mathsf{pk}}(C^{f*}; g^*)$
$K_h^* = \mathsf{H}(C^{f*}) \oplus K^*$
$T^* = \mathsf{F}(C^{f*}||Y^*||K_h^*||C^{e*})$
Return $\chi = (Y^*||K_h^*||C^{e*}||T^*)$

On Decryption-Query χ

$(Y||K_h||C^e||T) = \chi$
$C^f = \mathsf{Dec}_{\mathsf{sk}}(Y)$, $C = C^f||C^e$
$T' = \mathsf{F}(C^f||Y||K_h||C^e)$
$K = \mathsf{H}(C^f) \oplus K_h$
$M = S.Dec(K, C)$
if $T == T'$ **then**
  Return M
else
  Return $\perp$

On H-Query C^f

If $\exists\, h\ s.t.\ (C^f, h) \in I_{\mathsf{H}}$ **then** return h
$h \xleftarrow{\$} \{0,1\}^k$
$I_{\mathsf{H}} = I_{\mathsf{H}} \bigcup (C^f||h)$
return h;

On $\mathsf{H}^{\mathcal{A}}$-Query C^f

If $\exists\, h\ s.t.\ (C^f, h) \in I^{\mathcal{A}}_{\mathsf{H}}$ **then** return h
$h = \mathsf{H}(C^f)$
$I^{\mathcal{A}}_{\mathsf{H}} = I^{\mathcal{A}}_{\mathsf{H}} \bigcup \{(C^f, h)\}$
return v;

On F-Query $C^f||Y||K_h||C^e$

if $\exists\, T\ s.t.\ (C^f||Y||K_h||C^e, T) \in I_{\mathsf{F}}$ **then**
  return T
$T \xleftarrow{\$} \{0,1\}^k$
$I_{\mathsf{F}} = I_{\mathsf{F}} \bigcup \{(C^f||Y||K_h||C^e, T)\}$
return T;

On $\mathsf{F}^{\mathcal{A}}$-Query $C^f||Y||K_h||C^e$

if $\exists\, T\ s.t.\ (C^f||Y||K_h||C^e, T) \in I^{\mathcal{A}}_{\mathsf{F}}$ **then**
  return T
$T = \mathsf{F}(C^f||Y||K_h||C^e)$
$I^{\mathcal{A}}_{\mathsf{F}} = I^{\mathcal{A}}_{\mathsf{F}} \bigcup \{(C^f||Y||K_h||C^e, T)\}$
return T;

Figure 6.11: Game G0

Game G1: In Game G1: In decryption oracle, G1 adds a dummy event as $Flag_F \leftarrow new$, if the query to F oracle does not exist already in $I^{\mathcal{A}}_{\mathsf{F}}$, or $(C^f||Y||K_h||C^e, T) \notin I^{\mathcal{A}}_{\mathsf{F}}$. These changes are just conceptual. Therefore, $\Pr[Exp_{G1,\mathcal{A}}] = \Pr[Exp_{G0,\mathcal{A}}]$

Game G2: In Game G2, decryption oracle rejects the ciphertext if $Flag_F$ is new. Here difference between Game G1 and G2 arises when G2 rejects the ciphertext even when G1 has accepted them. This implies in G1, even on condition that $Flag_F$ is new, there is right matching of T and T'. It means that $\mathcal{A}$ knows an output value of the F function even without querying it. This could happen with probability of $\frac{q_d}{2^k}$, for q_d number of decryption queries. Therefore, $|\Pr[Exp_{G2,\mathcal{A}}] - \Pr[Exp_{G1,\mathcal{A}}]| \leq \frac{q_d}{2^k}$

Game G1 $\boxed{\textbf{G2}}$**:** Initialize $I_{\mathsf{F}} = I_{\mathsf{H}} = I_{\mathsf{F}}^{\mathcal{A}} = I_{\mathsf{H}}^{\mathcal{A}} = \emptyset$, $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^k)$. $Flag_F \in \{old, new\}$

On Decryption-Query χ

1 $(Y||K_h||C^e||T) = \chi$;
  $Flag_F \leftarrow old$
2 $C^f = \mathsf{Dec_{sk}}(Y)$, $C = C^f||C^e$
3 **If**$(C^f||Y||K_h||C^e, T) \notin I_{\mathsf{F}}^{\mathcal{A}}$ **then**
  $Flag_F \leftarrow new$ $\boxed{\text{Return } \perp}$
4 $T' = \mathsf{F}(C^f||Y||K_h||C^e)$
5 $K = \mathsf{H}(C^f) \oplus K_h$
6 $M = S.Dec(K, C)$
7 **If** $T == T'$ **then** Return M **else**
  Return $\perp$

Rest of oracles are same as G0

Figure 6.12: Game G1, G2: G1 adds dummy lines of code to G0, shown in dash box. G2 includes dummy lines of dash box and lines of code in solid line box

Game G3: Initialize $I_{\mathsf{F}} = I_{\mathsf{H}} = I_{\mathsf{F}}^{\mathcal{A}} = I_{\mathsf{H}}^{\mathcal{A}} = \emptyset$, $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^k)$.

On Decryption-Query χ

1 $(Y||K_h||C^e||T) = \chi$
2 **if** $(C^f||Y||K_h||C^e, T) \in I_{\mathsf{F}}^{\mathcal{A}}$ *s.t.*
  $E_{pk}(C^f; *) == Y$ **then**
    $K = \mathsf{H}(C^f) \oplus K_h$;
    $M = S.Dec(K, C)$; Return
    M
  else
    Return $\perp$

Rest of oracles are same as G0

Figure 6.13: Game G3: G3 output same as G2 without using sk and does not require $Flag_F$ anymore.

Game G2 and G3: G2 and G3 are same except that decryption oracle runs without using sk and decryption become independent from sk. For query $Y||K_h||C^e||T$, decryption oracle looks into $I_{\mathsf{F}}^{\mathcal{A}}$ for response T for a query $(C^f||Y||K_h||C^e)$ for some C^f. If $Y = \mathsf{Enc_{pk}}(C^f; g)$ for some g then decryption oracle proceeds forward otherwise return invalid. Condition $\mathsf{Enc_{pk}}(C^f; *) == Y$ is simulated using plaintext-checking oracle $\mathcal{O}^{PC}$.

$\Pr[Exp_{G3,\mathcal{A}}] = \Pr[Exp_{G2,\mathcal{A}}]$

Game G4 $\boxed{\textbf{G5}}$: Initialize $I_{\mathsf{F}} = I_{\mathsf{H}} = I_{\mathsf{F}}^{\mathcal{A}} = I_{\mathsf{H}}^{\mathcal{A}} = \emptyset$,, $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^k)$.

After Find Stage(AFS): $K^* \xleftarrow{\$} \{0,1\}^k$;$g^* \xleftarrow{\$}$ COINS;$\bar{K_h} \xleftarrow{\$} \{0,1\}^k$;$\bar{T} \xleftarrow{\$} \{0,1\}^k$

On Encryption-Query(M_d)

1 $C^* = S.Enc(K^*, M_d)$
2 $C^{f*}||C^{e*} = C^*$
3 $K_h^* = \mathsf{H}(C^{f*}) \oplus K^*$ $\boxed{K_h^* = \bar{K_h}}$
4 $Y^* = \mathsf{Enc_{pk}}(C^{f*}; g^*)$
5 $T^* = \mathsf{F}(C^{f*}||Y^*||K_h^*||C^{e*})$ $\boxed{T^* = \bar{T}}$
6 Return $\chi = (Y^*||K_h^*||C^{e*}||T^*)$

On $\mathsf{H}^{\mathcal{A}}$-Query C^f (*Adversary's query)

1 **If** $C^f == C^{f*}$ **then**
$Bad_f \leftarrow true$ $\boxed{\text{Halt}}$
2 **if** $\exists\ h\ s.t.\ (C^f, h) \in I_{\mathsf{H}}^{\mathcal{A}}$ **then**
return h
3 $h = \mathsf{H}(C^f)$
4 $I_{\mathsf{H}}^{\mathcal{A}} = I_{\mathsf{H}}^{\mathcal{A}} \bigcup \{(C^f, h)\}$
5 return v;

On $\mathsf{F}^{\mathcal{A}}$-Query $C^f||Y||K_h||C^e$ (*Adversary's query)

1 **If** $C^f == C^{f*}$ **then**
$Bad_f \leftarrow true$ $\boxed{\text{Halt}}$
2 **if** $\exists\ T\ s.t.\ (C^f||Y||K_h||C^e, T) \in I_{\mathsf{F}}^{\mathcal{A}}$ **then**
return T
3 $T = \mathsf{F}(C^f||Y||K_h||C^e)$
4 $I_{\mathsf{F}}^{\mathcal{A}} = I_{\mathsf{F}}^{\mathcal{A}} \bigcup \{(C^f||Y||K_h||C^e, T)\}$
5 return T;

Rest of oracles same as G3

Figure 6.14: Game G4 and G5: G4 adds dummy lines in dashed box compared to G3. G5 includes dummy lines and lines in solid line box.

Game G3 and G4: Game G4 and G3 are same, except a dummy event is added in $\mathsf{H}^{\mathcal{A}}$ and $\mathsf{F}^{\mathcal{A}}$ oracle in G4. This dummy event raises the event as $Bad_f \leftarrow true$ if adversary queries C^{f*} to $\mathsf{H}^{\mathcal{A}}$ or $C^{f*}||*$ to $\mathsf{F}^{\mathcal{A}}$. K^* and g^* are chosen randomly before the encryption query but after "find" stage (AFS). A dummy variable $\bar{K_h}$ is also chosen at random. These changes are just conceptual ones. Therefore, $Pr[Exp_{G4,\mathcal{A}}] = Pr[Exp_{G3,\mathcal{A}}]$.

Game G5: In Game G5, we halt the game if $Bad_f \leftarrow true$ happens. In case this event does not happens then K_h^* and T^* in encryption oracle can be replaced by any random string like $\bar{K_h}$ and $\bar{T}$ respectively. This change is then essentially

bounded by $(Pr[Bad_f] = Pr[(C^{f*}, *) \in I_{\mathsf{H}}^{\mathcal{A}} \vee (C^{f*}||*, *) \in I_{\mathsf{F}}^{\mathcal{A}}])$. Therefore, $|Pr[Exp_{G5,\mathcal{A}}] - Pr[Exp_{G4,\mathcal{A}}]| \leq Pr[Bad_f]$.
We continue games assuming halt has not happened. Later, we will calculate probability of Bad_f as a reduction to one-wayness of asymmetric encryption.

Game G6: In Game G6, some extra dummy random values $\bar{C} = \bar{C^f}||\bar{C^e} \in \{0,1\}^{Clen(M_d)}$ are chosen along with K^*, g^* and K_h^*. These changes are just dummy and conceptual therefore, $\Pr[Exp_{G6,\mathcal{A}}] = \Pr[Exp_{G5,\mathcal{A}}]$

Game G6 $\boxed{\textbf{G7}}$**:** Initialize $I_{\mathsf{F}} = I_{\mathsf{H}} = I_{\mathsf{F}}^{\mathcal{A}} = I_{\mathsf{H}}^{\mathcal{A}} = \emptyset$,, $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^k)$.
(AFS): $K^* \xleftarrow{\$} \{0,1\}^k$; $g^* \xleftarrow{\$}$ `COINS`; $K_h^* \xleftarrow{\$} \{0,1\}^k$; $T^* \xleftarrow{\$} \{0,1\}^k$;
$\bar{C} = \bar{C^f}||\bar{C^e} \xleftarrow{\$} \{0,1\}^{\mathsf{Clen}(M_d)}$

On Encryption-Query(M_d)	
1 $C^* = S.Enc(K^*, M_d)$ $\boxed{C^* = \bar{C}}$ 2 $C^{f*}||C^{e*} = C^*$ 3 $Y^* = \mathsf{Enc}_{\mathsf{pk}}(C^{f*}; g^*)$ 4 Return $\chi = (Y^*||K_h^*||C^{e*}||T^*)$	Rest of oracles same as G5

Figure 6.15: Game G6 and G7: G6 has some dummy change in code compare to G5 shown in dash box; this box contains some dummy variables and replacement of original values from random values. G7 includes lines of dash box and solid line box.

Game G6 and G7: From G6 to G7, $C^* = S.Enc(K^*, M_d)$ is replaced by a random string $\bar{C}$. This change is bounded by adversary $\mathcal{B}$ attacking the (one-time) encryption scheme S . Therefore, $|\Pr[Exp_{G7,\mathcal{A}}] - \Pr[Exp_{G6,\mathcal{A}}]| \leq Adv_{\mathcal{B},S}^{\$ot}$

Game G8: C^* is already a random string in G7 and independent from K^*, value of g^* can be chosen randomly independently from K^*. Now in G8, C^{f*} and C^{e*} are chosen randomly. All values for encryption oracle are computed randomly before-hand therefore Y^* is computed on C^{f*} and g^*. These changes bring no change of the view of $\mathcal{A}$ therefore are just conceptual. Therefore, $\Pr[Exp_{G8,\mathcal{A}}] = \Pr[Exp_{G7,\mathcal{A}}]$

Game G8: Initialize $I_{\mathsf{F}} = I_{\mathsf{H}} = I_{\mathsf{F}}^{\mathcal{A}} = I_{\mathsf{H}}^{\mathcal{A}} = \emptyset$, $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^k)$.
(AFS): $g^* \xleftarrow{\$} \mathtt{COINS}$; $K_h^* \xleftarrow{\$} \{0,1\}^k$; $T^* \xleftarrow{\$} \{0,1\}^k$ $C^{f*} \xleftarrow{\$} \{0,1\}^{\ell}$; $C^{*e} \xleftarrow{\$} \{0,1\}^{\mathsf{Clen}(M_d)-\ell}$; $Y^* = \mathsf{Enc}_{\mathsf{pk}}(C^{f*}; g^*)$

On Encryption-Query(M_d)

1 Return $\chi = (Y^* || K_h^* || C^{e*} || T^*)$

Rest of Oracles same as G7

Figure 6.16: Game G8: G8 has same output as G7 with simplified code and ciphertext is independent from d bit of M_d

Adversary C: Given $Y \xleftarrow{\$} \{0,1\}^{\ell + co_{pe}}$

Adversary $\mathcal{A}$: Initialize $I_{\mathsf{F}} = I_{\mathsf{H}} = I_{\mathsf{F}}^{\mathcal{A}} = I_{\mathsf{H}}^{\mathcal{A}} = \emptyset$, $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^k)$.
(AFS): $g^* \xleftarrow{\$} \{0,1\}^{Coins}$; $K_h^* \xleftarrow{\$} \{0,1\}^k$; $C^{e*} \xleftarrow{\$} \{0,1\}^{\mathsf{Clen}(M_d)-\ell}$; $T^* \xleftarrow{\$} \{0,1\}^k$; $Y^* = Y$

Rest of oracles same as G8

Finalization: If there exists a $(C^f, h) \in I_{\mathsf{H}}^{\mathcal{A}}$ or $(C^f || *, *) \in I_{\mathsf{F}}^{\mathcal{A}}$ such that $\mathcal{O}^{PC}(C^f, Y) = 1$ then return C^f;

Figure 6.17: Adversary $\mathcal{C}$ using $\mathcal{A}$, where $\mathcal{A}$ using oracle as per simulated in G8.

In G8, encryption oracle runs independent from d bit of M_d. And finally game become independent from sk and bit d. $Pr[Exp_{G8,\mathcal{A}}] = \frac{1}{2}$

G8 is the final game as $\mathcal{A}$ for $\mathcal{C}$ who tries to find pre-image of a given random Y using pk. Probability of Bad_f is equivalent to the probability of breaking one-wayness of asymmetric primitive Pe , which gives $Pr[Bad_f] \leq Succ_{\mathcal{C}_{\mathcal{A}},\mathsf{Pe}}^{\mathsf{OW}-PCA}$

This completes the proof.

□

6.4 Conclusion

To convert a fixed length OW-CPAasymmetric primitive (Pe) into CCA-secure PKE for arbitrary length messages using a padding scheme, a randomization of the input to Pe is required. This implies that there is a need for an extra output that enables to recover the used randomness during decryption. If we can process that randomness as early as possible during encryption and decryption, then real-time encryption could be achieved with a single pass.

This work describes a real-time CCA-secure asymmetric encryption scheme (REAL). We propose two version of scheme, choice of scheme depends upon choice of security assumption on asymmetric primitive as OW-CPA or OW-PCA. For trapdoor one-way permutation OW-CPA and OW-PCA are same. `REAL` allows to process/recover the randomness used at the beginning of encryption/decryption processes which helps in having real-time encryption and decryption. Usage of randomised one-time encryption schemes helps to reduce ciphertext-overhead; and OW-PCA security is found to be a stronger notion but allows to avoid re-encryption during decryption in case of probabilistic asymmetric encryption. Our generic conversion may be seen as the best alternative to other generic works when memory/bandwidth savings are a priority along with streaming high amount of data.

6.4.1 Subsequent scope

Methodology and versatility of Sponge structure can be seen in other different areas where padding scheme as message preprocessing is required. Current results of this work provides theoretical justifications, implementation of this work can provide more credibility to results from practical side. Using other primitives apart from Sponge permutation, like block ciphers and compression functions, and study their behavior under proposed setting is also an interesting line of work.

At this point, after achieving a generic CCA-secure encryption scheme framework, we move to an another area of work, signcryption, where OAEP type padding and CCA-secure encryption are required. Signcryption is found to be a more complex system to handle and prove its security because of extra functionalities and security requirements. Therefore, building a signcryption scheme from Sponge based padding can further spread the impact of this technique to another area of cryptography.

Chapter 7

Signcryption schemes with insider security in ideal permutation model

Contents

In this chapter, we introduce a Signcryption scheme using Sponge based message padding. First, we go through the background of signcryption, and its composition method in which message padding has a significant role. We discuss related works and elaborate common limitations in those works along with a comparison table to provide targeted motivation. We provide comparison results of our proposed signcryption scheme versus other schemes as part of the contribution. Following better results of Sponge based message padding in asymmetric encryption schemes, we describe our proposal of a Sponge based padding for signcryption scheme. First, we describe the scheme in message length restricted mode along with security proof and then using simple feed-forward operations we change the scheme into a more generic signcryption scheme for arbitrary long messages without compromising security. This two-step proposal helps in understanding of the limitations and security complexity of signcryption schemes under different assumptions on encryption and signature schemes.

7.1 Introduction

The aim of signcryption is to provide both confidentiality and authentication of messages more efficiently in a single routine than performing encryption and signing independently. The reduction of the computational cost makes signcryption more practical and it is a preferred option for e-commerce and e-mail applications, where both confidentiality and authentication are required. Zheng [103] introduced the signcryption notion in 1997. He proposed a signcryption solution that is based on El-Gamal [57] encryption and signature. As an open problem, Zheng [103] left the design of generic signcryption schemes that are not based on computationally intractable problems (such as RSA for instance).

The study of generic compositions of encryption and signature for constructing signcryption schemes has been initiated by An et al. [3]. They considered different generic methods for designing signcryption through a black-box composition of any secure signature and public-key encryption schemes. In particular, they showed that both "encrypt-then-sign" (EtS) and "sign-then-encrypt" (StE) lead to secure signcryption schemes. However, the parallel signcryption approach "sign-and-encrypt" (S&E) composition does not provide privacy since signature may reveal information about the encrypted messages. They introduced an alternative generic method termed as "commit-then-sign-and-encrypt" (CtS&E) that provides some security guarantee for S&E. Note that CtS&E compositions lead to parallel signcryption.

An et al. [3] also define two types of security on a signcryption scheme, namely, an outsider security and an insider security. The outsider security deals with an external adversary who knows the public keys of a sender and a receiver. In insider security model, attacks are coming from the other party that participates in the communication. In other words, an insider adversary is either the sender who wants to compromise the receiver confidentiality or the receiver who tries to defeat the sender unforgeability. Since security against an insider adversary implies security against an outsider adversary, the former is preferred.

A different security model for signcryption, which has been adopted in a few early papers [3, 49], is the two-user setting. In this model, a single sender interacts with a single receiver. However, as pointed out by Dent [49], security in the two-user model does not imply security in the multi-user model, in which either several senders communicate with the same receiver or alternatively, several receivers obtain messages from a single sender. Hence, to ensure realistic security concept, a multi-user security model must be adopted. The strongest security definitions, which captures both insider confidentiality and unforgeability for the multi-user setting, have been defined in [71]. For an overview of different security models, see [50, 73].

7.1.1 Background

In 2002, An et al. [3] presented a methodology to encrypt and sign in parallel. A plaintext m is first transformed into a pair (c, d) made of a commitment c and a de-commitment d in such a way that c reveals no information about m, while the pair (c, d) allows to recover m. Once the transformation $m \to (c, d)$ is done, the signer signs c and encrypts d in parallel using appropriate encryption and signature algorithms. On the receiver side, the signature on c is verified and d is recovered from its ciphertext. Both operations can be executed in parallel. Finally, the plaintext m is reconstructed from (c, d). Parallel execution of cryptographic algorithms decreases the computation time needed to signcrypt a message. It is equal to the maximum of either time required to encrypt or time needed to sign. The methodology also provides minimum security requirements from underlying encryption and signature algorithms. In two-user model, An et al. [3] claim that to provide generic chosen ciphertext (IND-gCCA) secure and existentially unforgeable (UF-CMA) signcryption, it is enough to use any IND-CCA secure encryption, UF-CMA secure signature scheme and a secure commitment scheme under CtS&E composition. This IND-gCCA security is weaker than IND-CCA secure encryption.

The work by An et al. [3] has instigated investigation into new ways to define signcryption schemes in more generic way. Note that early works present signcryption schemes whose security depends on specific intractable problems such as discrete logarithm (see [103]) and integer factoring (see [72, 100]). The authors of earlier works left an open question of designing signcryption under weaker security assumptions on encryption and signature schemes that do not relate to any specific intractability assumption. For example, the generic trapdoor one-wayness (OW) assumption is satisfied by the RSA encryption (when integer factorization is intractable) and the ElGamal encryption (when the computational Diffie-Hellman (CDH) problem is intractable). In this work, we consider cryptographic primitives (encryption and signature) whose security assumptions are generic.

Parallel signcryption is further investigated by Pieprzyk and Pointcheval [88]. They proposed to use a (2, 2)-Shamir secret sharing (SSS) as a commitment scheme. A plaintext m is first split into two shares (s_1, s_2), where any single share reveals no information about m. The first share s_1 is used as a commitment and signed while the second s_2 is encrypted. The authors of [88] proposed two version of their scheme. The first version, called generic parallel signcryption, provides IND-CCA and UF-CMA security for signcryption using any IND-CCA secure encryption and UF-CMA secure signature. This result is the same as the one obtained by [3]. The second version, called optimal parallel signcryption, applies an asymmetric padding OAEP [13]. This signcryption algorithm provides both IND-CCA and UF-CMA security in random oracle (RO) model assuming any OW encryption (such as the basic RSA) and any weakly secure signature (non-universally forgeable). Authors discuss the security of their schemes under insider security model in [89].

Dodis et al. [52, 53] propose a different approach to perform parallel signcryption. In their approach, they use a Feistel probabilistic padding, which can be viewed as a generalization of other existing probabilistic paddings such as OAEP, OAEP+, PSS-R, etc. These authors argue that their signcryption provides

IND-CCA and strong existential unforgeability (sUF-CMA) security assuming trapdoor one-way permutations only.

Hybrid signcryption is an attractive approach in design of signcryption schemes. It follows the idea of hybrid encryption [1,7,25,42,47,55,62,66]. Hybrid encryption consists of an asymmetric *key encapsulation mechanism* (KEM) and a symmetric *data encapsulation mechanism* (DEM). The first formal treatment of security of signcryption has been done by Dent (see [48, 49]). Some other related works can be found in [24, 36, 73, 101]. Converting a hybrid encryption scheme to a hybrid signcryption scheme turns out to be trickier than it looks. The main difficulty is an increased complexity of analysis that results from a more complex adversarial model. It is necessary to consider not only straight-forward attacks against authenticity and confidentiality of messages but also more intricate issues such as distinction between outsider and insider attacks. Moreover, CtS&E type compositions are always preferred as a base for constructing secure KEMs.

7.1.2 Limitation of Existing Schemes

A majority of signcryption schemes follow the sequential designs StE or EtS. Note that all schemes for hybrid signcryption with KEM/DEM [24, 36, 48, 49] follow the sequential design. This design limits the efficiency in a natural way. This limitation can be lifted easily by using the CtS&E composition method, which performs encryption and signing in parallel. Many signcryption schemes are built using some specific intractability assumptions (for example, intractability of discrete logarithm [6, 71, 103]). These constructions are not generic as the assumptions made limit the choice of underlying encryption and signature schemes. Constructions for hybrid signcryption are generic but they require stronger security properties from key encapsulation mechanisms (KEM) and data encapsulation mechanism (DEM). For example, a recent generic hybrid signcryption scheme given in [36] requires an IND-CCA secure key encapsulation mechanism, a one-time secure symmetric-key encryption, a one-time secure message authentication code and a strong existentially unforgeable signature scheme. These requirement are much stronger than those needed in already available non-hybrid schemes [88].

To the best of our knowledge, there is no hybrid signcryption scheme that claims IND-CCA security and existential unforgeability using weak security properties like one-wayness and universal-unforgeability. Most of the schemes require existential unforgeability on underlying signature scheme which is a stronger assumption than universal unforgeability. A common method used in works [52, 72, 88, 89] is an OAEP type padding. The padding gives rise to some common limitations such as: (1) it restricts message space, (2) it works with deterministic one-way encryption and deterministic signature only, and (3) it provides security in the random oracle (RO) model. Unavailability of different types of padding schemes limits the extension of work for the CtS&E composition. Table 7.1 gives a brief summary of generic signcryption schemes based on CtS&E.

In chapter 3, motivated by the OAEP design, we proposed another type of padding called SpAEP.

SpAEP is based on Sponge structure, where permutation is considered as ideal permutation, and has no restriction on maximum message space. Unlike KEM-DEM, the SpAEP padding provides an alternative by combining symmetric and asymmetric primitives without a strict delineation. In brief, SpAEP uses a versatile Sponge function and SpongeWrap [20–22] in pipelined fashion and its partial output is used as input to the asymmetric encryption scheme. This padding provides similar security guarantees as the OAEP padding but it is more efficient. The SpAEP padding can be used with trapdoor one-way permutations only.

7.1.3 Motivation

A randomised padding, like OAEP, is a powerful tool, which converts weakly secure fixed trapdoor one-way functions into public-key encryption that is secure against strong adaptively-chosen-ciphertext attacks. The padding has been used in signcryption as a part of the commitment scheme in the CtS&E composition. It is known that CtS&E allows the use of weak cryptographic primitives in generic way to achieve a strong security of signcryption. A good example of such composition are the results of [88, 89], which integrate any one-way encryption system (such as the basic RSA) with a weakly secure signature (non-universally forgeable signatures) into a strong chosen-ciphertext secure and existentially unforgeable signcryption in the RO model. The limitation of functionality like message space restriction or type of encryption scheme is inherited from the commitment or padding scheme used.

The Sponge-based padding proposed in [8] is versatile and has been used in a different security model for asymmetric encryption based on an ideal permutation. This padding scheme supports arbitrarily long messages, uses small domain

Schemes	Model	Encryption	Signature	Message length	# of other functions	Signcryption
An et al. [3]	No Specific	IND-CCA	UF-CMA	Restricted	Commitment scheme	IND-gCCA/UF-CMA
Pieprzyk et al. [88]	Random Oracle	OW-CPA	suUF-RMA	Restricted	3 hash, 1 Secret share scheme	IND-CCA/sUF-CMA
Dodis et al. [53] [52]	Random Oracle	OW-CPA	sUF-CMA	Restricted	1 Hash, 1 Commitment scheme	IND-CCA/sUF-CMA
				Unrestricted	1 Hash, 1 Commitment scheme, Symmetric encryption	
Our Result	Ideal Permutation	OW-CPA	suUF-RMA	Unrestricted	1 SpongeWrap, 1 Sponge Function ($\approxeq$2 Hash)	IND-CCA/UF-CMA
		OW-PCA	uUF-RMA			

Table 7.1: Generic Signcryption schemes Based on CtS&E type composition: "IND" stands for Indistinguishability, "OW" for One-wayness, "CPA/CMA" for Chosen plaintext/message attack, "CCA" for chosen ciphertext-attack, "UF" for existential unforgeability, "uUF" for universal unforgeability, "suUF" for strong uUF, and "RMA" for random message attack.
OW-CPA is more specific to trapdoor one-way permutation, OW-PCA is One-wayness under plaintext-checking attack.

permutations, and applies "on the fly" encryption. Its running time is equivalent to execution of a single Sponge function, which is equivalent to a hash function. Motivated by versatility of the Sponge-based padding and by "amplification" of security properties (as demonstrated in [88,89]), we would like to develop a generic signcryption scheme that is secure in the ideal permutation model. We intend to use weak asymmetric primitives such as trapdoor one-way encryption and universal unforgeable signature. The scheme is designed to support arbitrarily long messages.

7.2 Contributions

In this chapter, we make the following contributions:

1. We present a signcryption scheme in the ideal permutation model using Sponge structure. First, we propose a signcryption scheme for messages of a fixed length. Then we show how to extend the scheme so it works for arbitrarily long messages. Using simple tricks, we demonstrate how different combinations of probabilistic/deterministic encryption and signa-

ture schemes following weaker security requirements can be used without compromising security of scheme. To best of our knowledge, this is the first Sponge based signcryption scheme. We also believe that proposed signcryption scheme is the first scheme which also allows different combination of weakly secure encryption and signature schemes to yield a strong secure signcryption scheme along with support of arbitrarily long messages.

2. Security requirement for encryption is one-wayness and for signature scheme is universal unforgeability. These minimum security requirements are sufficient to achieve indistinguishability and existential unforgeability security against adaptive attacks. Such weak requirements were only fulfilled in [88, 89], but scope of [88, 89] is limited to fixed message space and deterministic encryption and signature schemes.

3. Apart from encryption and signature primitives, our scheme requires an ideal permutation only. The permutation we use is based on the well-known iterative Sponge structure. Note that after the success of KECCAK [23] in the SHA-3 competition [80], the Sponge structure is becoming more and more popular and can serve as a swiss army knife in cryptography.

4. Flexibility of the Sponge based padding allows to scale the system from relatively short messages to long ones while preserving security properties. Besides the complexity of security analysis does not increase. Note that the usage of extra redundant data in the Sponge padding plays a important role in supporting long messages.

The Sponge structure used for message padding resembles the padding proposed in chapter 3 and 5, but differs as follows. Some extra redundant data is used to allow usage of Sponge padding with signature to provide both unforgeability and confidentiality. $\mathsf{IND\text{-}CCA}$ security proof of $\mathsf{SignEnc}$ is similar to security proof of SpPad–Pe, but consider Ver also. Due to insider adversary model presence of Ver in $\mathsf{IND\text{-}CCA}$ proof is like public function known to adversary also.

Some properties are naturally inherited from Sponge based padding. Signcryption offers “on the fly” computation property during the signcryption and unsigncryption processes. Implementation require use of forward permutation only, which saves implementation effort and memory.

7.3 Sponge based padding for Signcryption

In this section, we provide Sponge based padding SpWrap, a modified SpAEP/Sp-Pad–Pe which is suitable for signcryption.

7.3.1 Description

Sponge based padding consist two functions: *SpWrap* and *Sponge*. *SpWrap* and *Sponge* take some of their length parameters from ENCRYPT and SIGN used in SIGNCRYPT.

SpWrap is based on an iterated ideal permutation $\pi : \{0,1\}^{(b=r+c)} \rightarrow \{0,1\}^b$ with an initial value IV. Function *SpWrap* is a tuple of two algorithm *SpWrap.Enc() and SpWrap.Dec()*.

On input message M from message space Msg$\subset \{0,1\}^*$. *SpWrap.Enc()* gives output $C||T$ using a random K from keyspace Key$\subset \{0,1\}^k$. *SpWrap.Enc()* takes input message M, $IV = IV_1||IV_2$, K and some length parameters like k, r, ℓ_{sg}. Output of *SpWrap.Enc()* is $C||T$ where $C > M$ and $\mid T \mid = k$. *SpWrap.Dec()* takes input a ciphertext $C||T$, $IV = IV_1||IV_2$, K and some length parameters like k, r, ℓ_{sg}. Output of *SpWrap.Dec()* is M or $\perp$.

SpWrap uses similar functioning of SpongeWrap [21], but its message padding is a little more specific than the general injective reversible padding used in the SpongeWrap. After applying injective-reversible padding to input message, which is required for smooth functioning of Sponge structure, we specifically add a 0^r-bit block before specific length ℓ_{sg}. This addition of extra block is required during parallel signcryption to prevent some trivial forgery attack which we will discuss later during the proof.

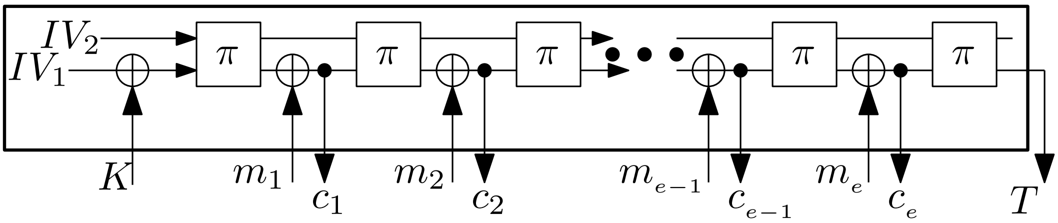


Figure 7.1: $SpWrap^{\pi}$: $SpWrap^{\pi}$ or simply $SpWrap : \{SpWrap.Enc, SpWrap.Dec\}$ is based on iterated permutation π. $SpWrap$ is similar to SpongeWrap function and works as authenticated encryption scheme. By default, Initial Value (IV) is considered as 0 (IV=0^b).
Sponge function: Shown figure can be viewed as Sponge function also by considering input $J = K||m_1||\ldots||m_e$, replace $IV_2 = 0^c$ from $IV_3 = 0^{c-1}||1$ and considering T as only output.

$\underline{SpWrap.Enc(K, M, IV_1||IV_2, r, k, \ell_{sg})}$:

1. $x = IV_1$; $w = IV_2$;
2. $\mathsf{checkin}(M, r, k, \ell_{sg})$=$m_1||\ldots||m_{(n+1)}$
3. $x = IV_1 \oplus 0^{(r-k)}||K$
4. **for** $i = 1 \to n+1$ **do**
   - $(x||w) = \pi(x||w)$
   - $x = x \oplus m_i$
   - $c_i = x$
5. $(x||w) = \pi(x||w)$; $T = \lfloor x \rfloor_k$
6. Return: $C||T = c_1||c_2||\ldots||c_{n+1}||T$

$\underline{SpWrap.Dec(K, C||T, IV_1||IV_2, r, k, \ell_{sg})}$:

1. $c_1||c_2||\ldots||c_{n+1}||T = C||T$ where each $| c_i |= r$
2. $x = IV_1 \oplus 0^{(r-k)}||K$; $w = IV_2$
3. **for** $i = 1 \to n+1$ **do**
   - $(x||w) = \pi(x||w)$
   - $m_i = x \oplus c_i$
   - $x = c_i$
4. $(x||w) = \pi(x||w)$; $T' = \lfloor x \rfloor_k$
5. $X' = m_1||\ldots||m_{n+1}$;
6. **if** $T == T'$ **then**
   - **If** $\exists\ M$ s.t. M =$\mathsf{checkout}(X', r, k, \ell_{sg})$ **then** Return:M **else** Return: $\perp$

   else
   - $\perp$

checkin(M, r, k, ℓ_{sg})

1. $X_1||X_2 = pad(M, r)$, where $| X_2 |= \ell_{sg} - r$
2. $X_1||0^r||X_2 = m_1||m_2|| \dots ||m_{n+1}$, where $|m_i| = r \ \forall 1 \leq i \leq (n+1)$ and $\exists \ m_i = 0^r$ such that $m_1|| \dots m_{i-1} = X_1$
3. return: $m_1||m_2|| \dots ||m_{n+1}$

checkout(X, r, k, ℓ_{sg})

1. **if** $\exists X_1, X_2$ *s.t.* $X_1||0^r||X_2 == X$, *where* $| X_2 |= \ell_{sg} - r$ **then**
   $X' = X_1||X_2$
   else
   Return $\perp$
2. Return: $unpad(X', r)$

$pad(x, r)$
$X =$
$x||1||0^{r-(|x|+1 \bmod r)-1}||1$
return X.

$unpad(y, r)$
{
if $\exists \ x \neq empty \ s.t. \ x||1||0^z||1 = y \ where \ 0 \leq z \leq r-1$ **then**
return x
else
return $\perp$
}

Sponge : *Sponge* which works exactly like Sponge function [20]. *Sponge* function has fixed b-bit initial value IV which is different from IV of SpongeWrap. In *Sponge*, we take $IV = IV_1||IV_3$ where $IV_3 = IV_2 \oplus 1$. *Sponge* takes $J \in \{0,1\}^*$ as input and output k-bit tag value h. We define the Sponge function based on π as follows:

$Sponge(IV_1||IV_3, J)$

1. $x||w = IV_1||IV_3$ where $|x| = r$
2. $j_1||j_2|| \dots ||j_n = pad(J, r)$, where $|j_i| = r \ \forall 1 \leq i \leq n$.
3. **for** $i = 1 \rightarrow n$ **do**
   $x = x \oplus j_i$
   $x||w = \pi(x||w)$
4. Return $\lfloor x \rfloor_k$

7.3.2 Properties

One useful property of *SpWrap* is its bijection property. Considering a fixed IV for *SpWrap*, each query to *SpWrap.Enc()* has a fixed chain of internal variables because of permutation π. Therefore, every different query will have its unique set of state values. No two different queries can have a similar whole set of state bits. First point of difference between two queries will create diversion in set values because of permutation π.

7.4 Parallel Signcryption: **SIGNCRYPT**

In this section, we describe our proposal of parallel signcryption using Sponge function based padding. To keep this scheme simple, we start with restricted message space and deterministic signature scheme. We remove these conditions of message space and signature scheme in Section 7.5.

7.4.1 Description

Building blocks of Parallel Signcryption SIGNCRYPT are:

- an encryption scheme ENCRYPT= (GenEnc, Enc, Dec),
- a signature scheme SIGN=(GenSig,Sign,Ver),
- a permutation $\pi : \{0,1\}^{(b=r+c)} \to \{0,1\}^b$ (assumed to behave like ideal permutation),
- For k-bit security of parallel signcryption, π having sufficient $r > c > k$ such that it should provide at-least k-bit security.
- We assume $\ell = n * r$ and $\ell_{sg} = m * r$ for some positive integers $n, m > 0$.
- A public function ID which maps public key of any user A to unique $\frac{r-k}{2}$-bit string in compatible string format as ID_A. Communicating party denoted as sender S and receiver R. This helps in describing of multi-users of system.
- Length of Message M is $\ell + \ell_{sg} - 2(k+1)$.

Key Generation: $\mathsf{Gen}(1^k) = \mathsf{GenSig} \times \mathsf{GenEnc}(1^k)$

Sender S generates $(\mathsf{sk}^{sig},\mathsf{pk}^{sig}) \leftarrow \mathsf{GenSig}_S(1^k)$ and
Receiver R generates $(\mathsf{sk}^{enc},\mathsf{pk}^{enc}) \leftarrow \mathsf{GenEnc}_R(1^k)$.
The sender keys are $(\mathsf{sk}_S,\mathsf{pk}_S)=(\mathsf{sk}^{sig},\mathsf{pk}^{sig})$ and

the receiver keys are (sk_R,pk_R)=(sk^{enc},pk^{enc}). Accordingly, SDK=(sk_S,sk_R) and VEK=(pk_S,pk_R). Using function ID, unique identities of sender S and receiver R will be ID_S and ID_R respectively.

Encrypt and Sign Algorithm: $\mathsf{SignEnc}_{\mathsf{SDK}_S,\mathsf{VEK}_R}(M)$

1. Compute $K||C||T = SpWrap.Enc(M, IV_1||IV_2, r, k, \ell_{sg})$, where $IV_1 = ID_S||ID_R$, $IV_2 = 0^c$, $|K| = k$ and r is input rate of π.
2. Parse $C||T$ into $S^1||S^2||T$, i.e. $C||T = S^1||S^2||T$, where $|S^1| = \ell$, $|S^2| = \ell_{sg}$.
3. Calculate (in parallel) Y_1=$\mathsf{Enc}_{\mathsf{pk}_R}(S^1)$, $\sigma = \mathsf{Sign}_{\mathsf{sk}_S}(S^2)$.
4. Calculate $K_h = K \oplus Sponge(S^1||Y_1)$, $T_k = T \oplus K$
5. The final output $(K_h, Y_1, Y_2 = (S^2, \sigma), T_k)$ is sent to receiver R.

Decrypt and Verify Algorithm: $\mathsf{VerDec}_{\mathsf{SDK}_R,\mathsf{VEK}_S}(K_h, Y_1, Y_2, T_k)$

1. Calculate (in parallel) $S^1 = \mathsf{Dec}_{\mathsf{sk}_R}(Y_1)$, $\top/\bot = \mathsf{Ver}_{\mathsf{pk}_S}(Y_2 = (S^2, \sigma))$. Ver returns either valid $\top$, or $\bot$ if signature is invalid. In case of return $\bot$, the decryption and verify algorithm VerDec returns $\bot$ and stops.
2. If Ver returns $\top$, then calculate $K = K_h \oplus Sponge(S^1||Y_1)$ and $T = T_k \oplus K$
3. Set $C = C^f||C^e = S^1||S^2$; also set $IV_1 = ID_S||ID_R, IV_2 = 0^c$.
4. Compute $M' = SpWrap.Dec(K||C||T, IV_1||IV_2, r, k, \ell_{sg})$ Return $M = M'$ if $M' \neq \bot$ else return $\bot$.

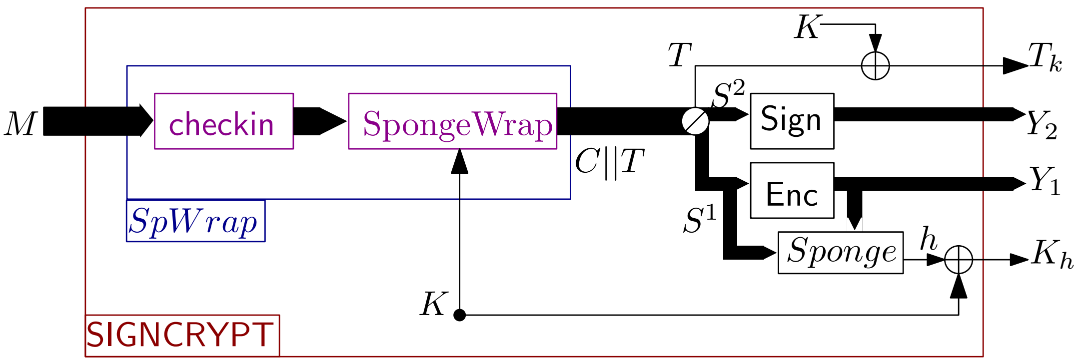


Figure 7.2: Sponge based Signcryption scheme SIGNCRYPT

Structural difference between SpPad–Pe and SIGNCRYPT : If we see closely, then overall structure of SIGNCRYPT and SpPad–Pe is similar except two things. One is obvious inclusion Sign. Second is change in padding formation through `checkin`, inclusion of a 0^r block in SpWrap just before giving the output

S^2 as input for Sign. This padding change is found to be crucial for insider secure support of Sign, which will be discussed as a part of proof.

Algorithm 7: Signcryption: $\mathsf{SignEnc}_{\mathsf{sk}_S,\mathsf{pk}_R}(M)$

Initialization: $x = IV_1 = 0^r$,
  $w = IV_2 = 0^c$, $IV_3 = IV_2 \oplus 1$,
Random Key: $K \xleftarrow{\$} \{0,1\}^k$;
$\mathsf{checkin}(M, r, k, \ell)$=$m_1||\ldots||m_{(n+1)}$
$x = ID_S||ID_R||K$
for $i = 1 \to n+1$ **do**
  $(x||w) = \pi(x||w)$
  $x = x \oplus m_i$
  $c_i = x$
$(x||w) = \pi(x||w)$; $T = \lfloor x \rfloor_k$
$(S^1)||(S^2) =$
  $(c_1||\ldots||c_e)||(c_{e+1}||\ldots||c_{n+1})$
$Y_1 = \mathsf{Enc}_{\mathsf{pk}_R}(S^1)$, $\sigma = \mathsf{Sign}_{\mathsf{sk}_S}(S^2)$
$pad(S^1||Y_1)$=$y_1||\ldots||y_j$;
  $x = IV_1; w = IV_3$
for $i = 1 \to j$ **do**
  $(x||w) = \pi((x \oplus y_i)||w)$
$K_h = \lfloor x \rfloor_k \oplus K$; $T_k = T \oplus K$
Return: $(K_h, Y_1, Y_2 = (S^2, \sigma), T_k)$

Algorithm 8: Unsigncryption: $\mathsf{VerDec}_{\mathsf{sk}_R,\mathsf{pk}_S}(K_h, Y_1, Y_2, T_k)$

Initialization: $IV_1 = 0^r$, $IV_2 = 0^c$,
  $IV_3 = IV_2 \oplus 1$,
$S^1 = \mathsf{Dec}_{\mathsf{sk}_R}(Y_1)$;
  $x = IV_1, w = IV_3$;
if $Ver_{pk_S}(Y_2 = (S^2, \sigma)) == \perp$ **then**
  Return $\perp$
$(c_1||\ldots||c_e)||(c_{e+1}||\ldots||c_{n+1}) =$
  $(S^1)||(S^2)$
$pad(S^1||Y_1)$=$y_1||\ldots||y_j$;
for $i = 1 \to j$ **do**
  $(x||w) = \pi((x \oplus y_i)||w)$
$K = \lfloor x \rfloor_k \oplus K_h$; $T = T_k \oplus K$
$x = ID_S||ID_R||K$; $w = IV_2$
for $i = 1 \to n+1$ **do**
  $(x||w) = \pi(x||w)$
  $m_i = x \oplus c_i$
  $x = c_i$
$(x||w) = \pi(x||w)$; $T' = \lfloor x \rfloor_k$
$X' = m_1||\ldots||m_{n+1}$;
if $T == T'$ **then**
  if $\exists\ M$ *s.t.* $M = \mathsf{checkout}(X', r, k, \ell)$ **then**
    Return:M
  else
    Return: $\perp$
else
  $\perp$

7.4.2 Security of Parallel Signcryption

Security of signcryption schemes is two fold, one about IND-CCA security and second is unforgeability under adaptive chosen message attack (UF-AdA). Before proceeding to detailed proof of each part individually, we provide a bird's eye view of each proof.

Theorem 8. *If the encryption scheme is* OW-PCA*, and the signature scheme is deterministic* uUF-RMA*, then the parallel signcryption scheme described in section 7.4 is secure*(IND/UF-AdA)*.*

Unforgeability

Proof Sketch: We are dealing with insider security model, the adversary has a target sender ID_S^* in mind and he/she knows the sender's public key pk_S^*. The adversary has access to the signcryption oracle under sk_S^*. Being working in multi-user setting, many receivers with different receiver ids are taken into consideration.

We make subsequent changes in permutation π such that π gives a permutation response for each new query but r bits out of b-bit output are random. Likewise, c bits out of b-bit output are always different for new input. The bound of these changes will be $\frac{(q_\pi-1)q_\pi}{2^{b+1}} + \frac{q_\pi(q_\pi+1)}{2^c}$ for q_π number of total queries on π. In an abstract way this bound include collision over b and c-bit output of π.

We start making changes in SignEnc oracle. We try to make output of SignEnc oracle as random output by using random output of π. We use the message-signature pair list Signlist having q_H elements, where messages are chosen at random and signature are calculated based on sk_{S^*}. Because we are working in multi-user security model, SignEnc accepts different receiver id's along with M. Finally SignEnc can respond with random output of π and using pre-computed Signlist, likewise independent of $\mathsf{Sign}_{\mathsf{sk}_{S^*}}$. The bound of changing original response with random response comes out to be $q_{sc} \cdot \frac{q_\pi^{\mathcal{A}}}{2^k}$. This bound captures the probability of guessing used randomness K during q_{sc} number of signcryption queries.

We modify VerDec oracle such that, we detect existential forgery on VerDec and show a reduction to universal forgery on Ver. Whenever we discuss a forgery we consider $ID_R = ID_R^*$ in VerDec given by adversary with target signcryptext y^* and related pk_{R^*} and sk_{R^*} for target sender ID_{S^*}. For detecting a valid forgery, we cross check set I_{vd}, consist input-output of π during unsigncryption, against a set $I_\pi^{\mathcal{A}}$ and I_π^{sc} consist of input-output of π maintained by adversary and signcryption oracle respectively. Let q_{usc} be number of unsigncryption queries and q_{sc} be number of signcryption queries. We show that if $I_{vd} \subset I_\pi^{sc}$, then this is not an existential forgery. We show if $(I_{vd} \not\subset I_\pi^{sc}$ && $I_{vd} \not\subset I_\pi^{\mathcal{A}})$ *or* $I_{vd} \subseteq I_\pi^{\mathcal{A}}$ then probability of having an existential forgery is negligible. The bound for these changes comes out to be $\frac{q_{usc}}{2^k} + \frac{q_{sc}}{2^r} + Adv_{\mathrm{SIGN}}^{\textsf{uuf-rma}}(k)$. This bound capture the probability of producing a target collision on T or target collision on input of Ver or creating a valid signature on random input of Sign.

During unforgeability proof this is natural to assume that encryption scheme is following trapdoor one-wayness and its correctness condition.

Following lemma can be derived from Theorem 8:

Lemma 1. *If there exists an adversary $\mathcal{A}$ against the* UF-AdA *security of the parallel signcryption scheme with advantage $Adv_{\textsf{SignEnc}}^{\textsf{uf-ada}}(k)$ whose running time is bounded by t and who makes at most $q_\pi^{\mathcal{A}}$ queries to the permutation $\pi : \{0,1\}^{b=r+c} \to \{0,1\}^b$ and q_{sc} queries to the signcryption oracle and q_{usc} queries to the unsigncryption oracle. Then there exists an attacker $\mathcal{B}$ against the* uUF-RMA *security of the signature scheme with advantage $Adv_{\mathrm{SIGN}}^{uUF-RMA}(k)$ whose running time is bounded by $t' \geq t + q_{sc}(\tau + O(1))$, where τ denotes the maximal running time of the encryption and signing algorithm, for which*
$Adv_{Signcrypt}^{\textsf{uf-ada}}(k) \leq Adv_{\mathrm{SIGN}}^{\textsf{uuf-rma}}(k) + \frac{(q_\pi-1)q_\pi}{2^{b+1}} + \frac{q_\pi(q_\pi+1)}{2^c} + q_{sc} \cdot \frac{q_\pi^{\mathcal{A}}}{2^k} + \frac{q_{usc}}{2^k} + \frac{q_{sc}}{2^r},$

where q_π is total number of π queries including queries by adversary, signcryption and unsigncryption oracle.

Proof. We consider the similar experiment of UF-AdA as described in section 2.3. We follow, following experiment for UF-AdA experiment for SIGNCRYPT by Adversary $\mathcal{A}$:

$Exp_{\mathsf{SIGNCRYPT},\mathcal{A}}^{UF-AdA}(k)$:

1. $(\mathsf{sk}_{S^*},\mathsf{pk}_{S^*}) \leftarrow \mathsf{GenSig}_{S^*}(1^k)$
2. $(y^*, ID_{R^*}) \leftarrow \mathcal{A}^{\mathsf{SignEnc}_{\mathsf{sk}_{S^*}}(\cdot,\cdot),\pi(\cdot)}(\mathsf{pk}_{S^*})$
3. Mapping pk_{R^*} using ID_{R^*}, where $(\mathsf{sk}_{R^*},\mathsf{pk}_{R^*}) \leftarrow \mathsf{GenEnc}_{R^*}(1^k)$
4. $M^* \leftarrow \mathsf{VerDec}(\mathsf{pk}_{R^*}, \mathsf{sk}_{S^*}, y^*)$
5. **if** $M^* \neq \perp$ and (M^*, ID_{R^*}) query never made to $\textit{SignEnc}_{\mathsf{sk}_{S^*}}(\cdot,\cdot)$ oracle **then**
   Return 1

else

∟ Return 0

Advantage of adversary $\mathcal{A}$ is given by following probability:

$Adv^{\textsf{uf-ada}}_{Signcrypt}(k) = \Pr[Exp^{UF-AdA}_{\textsf{SIGNCRYPT},\mathcal{A}}(k) = 1]$

We use game based proof framework [16]. We are dealing with insider security model, the adversary has a target sender ID^*_S in mind and he/she knows the sender public key pk^*_S. The adversary has access to the signcryption oracle under sk^*_S. We denote the winning event of forging a signcryptext in Game **Gi** by G_i.

Game G0 represent original Signcryption game for UF-AdA. Adversary issues q_{sc} number of queries on Signcryption oracle specifying Receiver ID_R in each query using ID^*_S. Adversary $\mathcal{A}$'s target is to give a target ID^*_R and signed ciphertext $(K^*_h, Y^*_1, Y^*_2 = (S^{2*}, \sigma^*), T^*_k)$, such that $\textsf{VerDec}_{\textsf{sk}^*_R,\textsf{pk}^*_S}(K^*_h, Y^*_1, Y^*_2, T^*_k) = M^* \neq \perp$ where (M^*, ID^*_R) should not be queried by $\mathcal{A}$ to SignEnc. $\mathcal{A}$ might ask (M, ID^*_R) or (M^*, ID_R) to SignEnc. Therefore,

$$\Pr[G_0] = \Pr[Exp^{UF-AdA}_{G0,\mathcal{A}}(k) = 1] = \Pr[Exp^{UF-AdA}_{\textsf{SIGNCRYPT},\mathcal{A}}(k) = 1]|$$

From Game **G0** to Game **G4**, we make successive changes in permutation π. Modified π gives a permutation response for each new query such that r bits out of b-bit output are random. Likewise, c bits out of b-bit output are always different for new input. This helps us to exploit the permutation property of Sponge and make an output C deterministic for a specific input K, M and IV. Any change in any one of the four values (C, M, K, IV) will make at-least one value random. Here "any change" implies, while establishing relation between (C, M, K, IV), if any input-output pair of π is not defined already then essentially one of the part is new or randomly generated.

Game G0: Initialize $I_\pi = I_\pi^{\mathcal{A}} = \emptyset$, $IV_1 = 0^r$ $IV_2 = 0^c$, $IV_3 = IV_2 \oplus 1$, $(\mathsf{sk}_R,\mathsf{pk}_R) \leftarrow \mathsf{GenEnc}(1^k)$, pk_S^*, ID_S^*; $\mid ID \mid \in \{0,1\}^{(r-k)/2}$
`Signlist` : $\{(S_i,\sigma_i) : \sigma_i = \mathsf{Sign}_{\mathsf{sk}_S^*}(S_i)\ \forall 1 \le i \le q_H$ and each S_i chosen randomly$\}$.

On SignEnc-Query M, ID_R

1. $K \xleftarrow{\$} \{0,1\}^k$; $x = IV_1$; $w = IV_2$;
2. $\mathsf{checkin}(M,r,k,\ell_{sg}) = m_1||\ldots||m_{(n+1)}$
3. $x = ID_S^*||ID_R||K$
4. **for** $i = 1 \to n+1$ **do**
   - $(x||w) = \pi(x||w)$
   - $x = x \oplus m_i$
   - $c_i = x$
5. $(x||w) = \pi(x||w)$; $T = \lfloor x \rfloor_k$
6. $S^1||S^2||T =$
   $c_1||\ldots||c_e||c_{e+1}||\ldots||c_{n+1}||T$
7. $Y_1 = \mathsf{Enc}_{\mathsf{pk}_R}(S^1)$, $\sigma = \mathsf{Sign}_{\mathsf{sk}_S^*}(S^2)$
8. $pad(S^1||Y_1) = y_1||\ldots||y_j$;
   $x = IV_1; w = IV_3$
9. **for** $i = 1 \to j$ **do**
   - $(x||w) = \pi(x \oplus y_i||w)$
10. $K_h = \lfloor x \rfloor_k \oplus K$; $T_k = T \oplus K$
11. Return: $(K_h, Y_1, Y_2 = (S^2,\sigma), T_k)$

On VerDec-Query (K_h, Y_1, Y_2, T_k)

1. $S^1 = \mathsf{Dec}_{\mathsf{sk}_R}(Y_1)$; $x = IV_1, w = IV_3$;
2. **if** $\mathit{Ver}_{\mathsf{pk}_S^*}(Y_2 = (S^2,\sigma)) == \perp$ **then**
   - Return $\perp$
3. $c_1||\ldots||c_e||c_{e+1}||\ldots||c_{n+1} = S^1||S^2$
4. $pad(S^1||Y_1) = y_1||\ldots||y_j$;
5. **for** $i = 1 \to j$ **do**
   - $(x||w) = \pi(x \oplus y_i||w)$
6. $K = \lfloor x \rfloor_k \oplus K_h$; $T = T_k \oplus K$
7. $x = ID_S^*||ID_R||K$; $w = IV_2$
8. **for** $i = 1 \to n+1$ **do**
   - $(x||w) = \pi(x||w)$
   - $m_i = x \oplus c_i$
   - $x = c_i$
9. $(x||w) = \pi(x||w)$; $T' = \lfloor x \rfloor_k$
10. $X' = m_1||\ldots||m_{n+1}$;
11. **if** $T == T'$ **then**
    - **if** $\exists\ M$ *s.t.* $M = \mathsf{checkout}(X', r, k, \ell_{sg})$ **then**
      - Return:M
    - **else**
      - Return: $\perp$

    else
    - $\perp$

On π-Query m, where $m \in \{0,1\}^b$

1. let $(x||w) = m$, where $x \in \{0,1\}^r$, $w \in \{0,1\}^c$,
2. if $(m,v) \in I_\pi$ then return v
3. $v \xleftarrow{\$} \{0,1\}^b$
4. if $\exists\ m'$ s.t $(m',v) \in I_\pi$, then $v \xleftarrow{\$} \{0,1\}^b \setminus \{v : (*,v) \in I_\pi\}$, where $* \in \{0,1\}^b$
5. $I_\pi = I_\pi \bigcup \{(m,v)\}$
6. return v;

On π^{-1}-Query v, where $v \in \{0,1\}^b$

1. if $(m,v) \in I_\pi$ then return m
2. $m \xleftarrow{\$} \{0,1\}^b$
3. if $\exists\ v'$ s.t $(m,v') \in I_\pi$, then $m \xleftarrow{\$} \{0,1\}^b \setminus \{m : (m,*) \in I_\pi\}$, where $* \in \{0,1\}^b$
4. $I_\pi = I_\pi \bigcup \{(m,v)\}$
5. return m;

On $\pi_{\mathcal{A}}$-Query m, where $m \in \{0,1\}^b$

1. $v = \pi(m)$
2. $I_\pi^{\mathcal{A}} = I_\pi^{\mathcal{A}} \bigcup \{(m,v)\}$
3. return v;

On $\pi_{\mathcal{A}}^{-1}$-Query v, where $v \in \{0,1\}^b$

1. $m = \pi^{-1}(v)$
2. $I_\pi^{\mathcal{A}} = I_\pi^{\mathcal{A}} \bigcup \{(m,v)\}$
3. return m;

Figure 7.3: Game G0

Game $\boxed{\textbf{G1}}$ **and G2:** Initialize $I_\pi = I_\pi^{\mathcal{A}} = \emptyset$, $IV_1 = 0^r$ $IV_2 = 0^c$, $IV_3 = IV_2 \oplus 1$, $(\mathsf{sk}_R, \mathsf{pk}_R) \leftarrow \mathsf{GenEnc}(1^k)$, pk_S^*, ID_S^*;
`Signlist` : $\{(S_i, \sigma_i) : \sigma_i = \mathsf{Sign}_{\mathsf{sk}_S^*}(S_i)\ \forall 1 \leq i \leq q_H$ and each S_i chosen randomly$\}$.

On π-Query m, where $m \in \{0,1\}^b$

1. let $(x||w)$=m, where $x \in \{0,1\}^r$, $w \in \{0,1\}^c$,
2. if $(m, v) \in I_\pi$ then return v
3. $v \xleftarrow{\$} \{0,1\}^b$
4. if $\exists$ m' s.t $(m', v) \in I_\pi$, then **bad**$\leftarrow$**true** and $\boxed{v \xleftarrow{\$} \{0,1\}^b \setminus \{v : (*, v) \in I_\pi\}}$, where $* \in \{0,1\}^b$
5. $I_\pi = I_\pi \bigcup \{(m, v)\}$
6. return v;

On π^{-1}-Query v, where $v \in \{0,1\}^b$

1. let $(v_1||v_2)$=m, where $v_1 \in \{0,1\}^r$, $v_2 \in \{0,1\}^c$,
2. if $(m, v) \in I_\pi$ then return m
3. $m \xleftarrow{\$} \{0,1\}^b$
4. if $\exists$ v' s.t $(m, v') \in I_\pi$, then **bad**$\leftarrow$**true** and $\boxed{m \xleftarrow{\$} \{0,1\}^b \setminus \{m : (m, *) \in I_\pi\}}$, where $* \in \{0,1\}^b$
5. $I_\pi = I_\pi \bigcup \{(m, v)\}$
6. return m;

Rest of oracles are same as G0

Figure 7.4: Game G1 and Game G2: Dash-box has dummy line of code, with respect to G0, added and shared with both G1 and G2. G1 is with solid-box and G2 is without solid-box.

Game G1 and G2: We start making changes in permutation π. In **G1**, we take response of π randomly and differently from previous responses using set I_π. In **G2**, π queries simulates as random function that is for every new input, output is random, need not to be different. So in G2 π gives random response without cross checking it in previous input-output response list I_π. G1 and G2 remains identical until output of π query collides with any of the previous outputs. This collision is denoted as *bad* event. Probability that random response chosen as output of π will collide with any previous response is $\frac{(q_\pi - 1)q_\pi}{2^{b+1}}$, where q_π is total number of queries on π (and π^{-1}) either from oracle calls by different oracle or by adversary $\mathcal{A}$. Therefore, $|\Pr[G_2] - \Pr[G_1]| \leq \frac{(q_\pi - 1)q_\pi}{2^{b+1}}$

Game G3 and G4: Game **G3** remains same as **G2**. In **G3**, we split up output v of π in input-rate v_1 and capacity-rate v_2. We also have a set $\mathcal{L}_c$ initially having element of value IV_2 and IV_3. Output v of π is chosen at random from previous outputs. We mark an event as *bad*$\leftarrow$ *true* in case v_2 is part of any previous output; $v_2 \in \mathcal{L}_c$. In **G4**, π converted back to permutation from random function. Now, In G4, if *bad* $\leftarrow$ *true* happens then v_2 is chosen again randomly from its set but rejecting the values already in set $\mathcal{L}_c$.

So, in case of *bad* $\leftarrow$ *true*, input-rate part of π's output at random and capacity-part differently from all previous capacity-parts of outputs. In G4, π works again as ideal permutation but permutation is happening over capacity-parts of output. After every query, set I_π and $\mathcal{L}_c$ are updated in accordance to input-output response of π. Probability of $bad \leftarrow true$ will be $\frac{q_\pi(q_\pi+1)}{2^c}$. Therefore, $|\Pr[G_4] - \Pr[G_3]| \leq \frac{q_\pi(q_\pi+1)}{2^c}$

Game G3 and Game $\boxed{\textbf{G4}}$: Initialize $I_\pi = I_\pi^{\mathcal{A}} = \emptyset$, $IV_1 = 0^r$ $IV_2 = 0^c$, $IV_3 = IV_2 \oplus 1$, $\mathcal{L}_c = \{IV_2, IV_3\}$, $(\mathsf{sk}_R,\mathsf{pk}_R) \leftarrow \mathsf{GenEnc}(1^k)$, pk_S^*, ID_S^*;
`Signlist` : $\{(S_i, \sigma_i) : \sigma_i = \mathsf{Sign}_{\mathsf{sk}_S}(S_i)\ \forall 1 \leq i \leq q_H$ and each S_i chosen randomly$\}$.

On π-Query m, where $m \in \{0,1\}^b$

1. let $(x||w)$=m,where $x \in \{0,1\}^r$, $w \in \{0,1\}^c$,
2. if $(m, v) \in I_\pi$ then return v
3. $v_1||v_2 \xleftarrow{\$} \{0,1\}^b$,where $v_1 \in \{0,1\}^r$, $v_2 \in \{0,1\}^c$,
4. if $v_2 \in \mathcal{L}_c \bigcup \{w\}$, then **bad**$\leftarrow$**true** and $\boxed{v_2 \xleftarrow{\$} \{0,1\}^c \setminus \mathcal{L}_c \bigcup \{w\}}$
5. $I_\pi = I_\pi \bigcup \{(m, v_1||v_2)\}$ and $\mathcal{L}_c = \mathcal{L}_c \bigcup \{v_2, w\}$
6. return $v = v_1||v_2$;

On π^{-1}-Query v. where $v \in \{0,1\}^b$

1. let $(v_1||v_2)$=v,where $v_1 \in \{0,1\}^r$, $v_2 \in \{0,1\}^c$,
2. if $(m, v) \in I_\pi$ then return m
3. $m'||m'' \xleftarrow{\$} \{0,1\}^b$,where $m' \in \{0,1\}^r$, $m'' \in \{0,1\}^c$,
4. if $m'' \in \mathcal{L}_c \bigcup \{v_2\}$, then **bad**$\leftarrow$**true** and $\boxed{m'' \xleftarrow{\$} \{0,1\}^c \setminus \mathcal{L}_c \bigcup \{v_2\}}$
5. $I_\pi = I_\pi \bigcup \{(m'||m'', v)\}$ and $\mathcal{L}_c = \mathcal{L}_c \bigcup \{m'', v_2\}$
6. return $m = m'||m''$;

Rest Oracles are same as G0

Figure 7.5: Game G3 and Game G4: Dash box shows dummy line of code added in G3 with respect to G2. G3 is without solid-box and G4 with solid-box.

From Game **G5** to **G9**, we start making changes in $\mathsf{SignEnc}$ oracle. We try to make output of $\mathsf{SignEnc}$ oracle as random output by using random output of π. We use the message-signature pair list $\mathsf{Signlist}$ having q_H elements, where message are chosen at random and signature are calculated based on sk_{S^*}. In last $\mathsf{SignEnc}$ can respond random output using pre-computed $\mathsf{Signlist}$, likewise independent of $\mathsf{Sign}_{\mathsf{sk}_{S^*}}$.

Game G5, G6: G5 is same as G6 with no change. In Game **G6**, in $\mathsf{SignEnc}$ we add a dummy random string $C^*||T^*$ equivalent length of $C||T$, shown as dash-box. Game **G5** and **G6** are same except some dummy lines are added in

G6 at step 4,5 in SignEnc. In these dummy lines, a random $C^*||T^*$ is chosen at random and C^* is spitted into c_i^* where $1 \le i \le n+1$ and each $|c_i^*| = r$

Game G5 and $\boxed{\textbf{G6}}$ Initialize Same as G4

On SignEnc-Query M, ID_R

1. $K \xleftarrow{\$} \{0,1\}^k$;
   $x = IV_1 = ID_S^*||ID_R||0^k$; $w = IV_2$;
2. $\mathsf{checkin}(M, r, k, \ell_{sg}) = m_1|| \ldots ||m_{(n+1)}$
3. $x = ID_S^*||ID_R||K$
4. $\boxed{C^*||T^* \xleftarrow{\$} \{0,1\}^{((n+1)*r)+r}}$
5. $\boxed{c_1^*||c_2^*|| \ldots ||c_{n+1}^* = C^*; T^* = \lfloor T^* \rfloor_k}$
6. **for** $i = 1 \to (n+1)$ **do**
   $(x||w) = \pi(x||w)$
   $x = x \oplus m_i$
   $c_i = x$
7. $(x||w) = \pi(x||w)$; $T = \lfloor x \rfloor_k$
8. $S^1||S^2||T =$
   $c_1|| \ldots ||c_e||c_{e+1}|| \ldots ||c_{n+1}||T$
9. $Y_1 = \mathsf{Enc}_{\mathsf{pk}_R}(S^1)$; $\sigma = \mathsf{Sign}_{\mathsf{sk}_S^*}(S^2)$;
10. $pad(S^1||Y_1) = y_1|| \ldots ||y_j$;
    $x = IV_1; w = IV_3$
11. **for** $i = 1 \to j$ **do**
    $(x||w) = \pi(x \oplus y_i||w)$
12. $K_h = \lfloor x \rfloor_k \oplus K$; $T_k = T \oplus K$
13. Return: $(K_h, Y_1, Y_2 = (S^2, \sigma), T_k)$

On π-Query m, where $m \in \{0,1\}^b$

1. let $(x||w)$=m,where $x \in \{0,1\}^r$,
   $w \in \{0,1\}^c$,
2. if $(m, v) \in I_\pi$ then return v
3. $v_1||v_2 \xleftarrow{\$} \{0,1\}^b$,where $v_1 \in \{0,1\}^r$,
   $v_2 \in \{0,1\}^c$,
4. if $v_2 \in \mathcal{L}_c \bigcup \{w\}$, then
   $v_2 \xleftarrow{\$} \{0,1\}^c \setminus \mathcal{L}_c \bigcup \{w\}$
5. $I_\pi = I_\pi \bigcup \{(m, v_1||v_2)\}$ and
   $\mathcal{L}_c = \mathcal{L}_c \bigcup \{v_2, w\}$
6. return $v = v_1||v_2$;

On π^{-1}-Query v. where $v \in \{0,1\}^b$

1. let $(v_1||v_2)$=v,where
   $v_1 \in \{0,1\}^r$, $v_2 \in \{0,1\}^c$,
2. if $(m, v) \in I_\pi$ then return m
3. $m'||m'' \xleftarrow{\$} \{0,1\}^b$,where $m' \in \{0,1\}^r$,
   $m'' \in \{0,1\}^c$
4. if $m'' \in \mathcal{L}_c \bigcup \{v_2\}$, then
   $m'' \xleftarrow{\$} \{0,1\}^c \setminus \mathcal{L}_c \bigcup \{v_2\}$
5. $I_\pi = I_\pi \bigcup \{(m'||m'', v)\}$ and
   $\mathcal{L}_c = \mathcal{L}_c \bigcup \{m'', v_2\}$
6. return $m = m'||m''$;

Rest of Oracles same same as G0

Figure 7.6: Game G5 and G6: dash box shows added dummy line of codes in G6 compare to G5. G5 has same code as G4.

Game G6, G7: In **G7**, we change response of π in accordance to c_i^*, such that SpPad.Enc output $C^*||T^*$ for M on K. As we already know, r-bit part of b-bit output of π is random. Therefore, we can replace the random output x of π with another random value $m_i \oplus c_i$. Such a change will produce $C^*||T^*$ as output response of π from its "for" loop. Now, $S^1||S^2||T = C^*||T^*$ and this is used for calculating encryption and signature for final output. Here, $C^* = c_1|| \ldots ||c_e||c_{e+1}|| \ldots ||c_{n+1}$, $S^1 = c_1|| \ldots ||c_e$ and $S^2 = c_{e+1}|| \ldots ||c_{n+1}$. We store input-output response of π, called in SignEnc, in a set I_π^{sc}.

This change of response might get failed if response of fist π call using K in SignEnc, is already defined by $\mathcal{A}$ query in $I_\pi^{\mathcal{A}}$ using $\pi^{\mathcal{A}}$ and public known IV_2 and IDs. Because if first response of π using K in SignEnc goes collision free then all successive response will be new due to permutation property. Therefore, probability of failure of this response change in G7 for q_{sc} number of queries is $\frac{q_\pi^{\mathcal{A}}}{2^k}$. Therefore, $|\Pr[G_7] - \Pr[G_6]| \leq q_{sc} \cdot \frac{q_\pi^{\mathcal{A}}}{2^k}$.

Game $\boxed{\textbf{G6}}$, (G7) Initialize Same as G4

On SignEnc-Query M, ID_R

1. $K \xleftarrow{\$} \{0,1\}^k$; $x = IV_1 = 0^k$; $w = IV_2$;
2. $\mathsf{checkin}(M, r, k, \ell_{sg})$=$m_1||\ldots||m_{(n+1)}$
3. $x = ID_S^*||ID_R||K$
4. $C^*||T_x^* \xleftarrow{\$} \{0,1\}^{((n+1)*r)+r}$
5. $c_1^*||c_2^*||\ldots||c_{n+1}^* = C^*$; $T^* = \lfloor T_x^* \rfloor_k$
6. **for** $i = 1 \to (n+1)$ **do**
   - $v = x||w$;
   - $\boxed{(x||w) = \pi(x||w)}$ $((x = c_i^* \oplus m_i)||w) = \pi(x||w)$;
   - $v' = x||w$; $I_\pi^{sc} = I_\pi^{sc} \bigcup \{v, v'\}$
   - $x = x \oplus m_i$
   - $c_i = x$
7. $\boxed{(x||w) = \pi(x||w); T = \lfloor x \rfloor_k;}$ $((T_x^*)||w) = \pi(x||w)$; $T^* = \lfloor T_x^* \rfloor_k$;
8. $S^1||S^2||T = c_1||\ldots||c_e||c_{e+1}\ldots||c_{n+1}||T$
9. $S^1||S^2||T = c_1^*||\ldots||c_e^*||c_{e+1}^*||\ldots||c_{n+1}^*||T^*$
10. $Y_1 = \mathsf{Enc}_{\mathsf{pk}_R}(S^1)$; $\sigma = \mathsf{Sign}_{\mathsf{sk}_S^*}(S^2)$;
11. $pad(S^1||Y_1)$=$y_1||\ldots||y_j$; $x = IV_1; w = IV_3$
12. **for** $i = 1 \to j$ **do**
   - $(x||w) = \pi(x \oplus y_i||w)$
13. $K_h = \lfloor x \rfloor_k \oplus K$; $T_k = T \oplus K$
14. Return: $(K_h, Y_1, Y_2 = (S^2, \sigma), T_k)$

Rest of Oracles are same as G5

Figure 7.7: Game G6 and G7: G6 follows the code without oval-box, G7 follows the code without solid box.

Game G7, G8: In **G8**, we chose a new message-signature pair from `Signlist` at random. We replace the chosen message S_i from `Signlist` with S^2 of π loop's

(SpPad) output. In **G8**, before start calculating SpPad and after generating $C^*||T^*$, we set $S^1||S^2||T = C^*||T^*$. Then we replace S^2 with S_i of (message-signature) pair list and then again set $C^*||T^* = S^1||S^2||T$. Rest code remain same like **G7**. Here we replace random S^2 with a random S_i of Signlist and calculating rest same as **G7**. Because both S_i and S^2 are random, therefore there is no difference will arise in Game **G7** and **G8**.

Game G8, G9: In **G9**, code remain same like **G8**, instead to calculate $\sigma = \mathsf{Sign}_{\mathsf{sk}_S}(S^2 = S_i)$ one can simple replace this operation with pre-calculated σ_i for S_i from `Signlist`. Now, SignEnc is independent of sk_S of Sign and later available to Adversary $\mathcal{B}$ for uUF-RMA attack on Sign.

Game G8, $\boxed{\text{G9}}$ Initialize Same as G4

On SignEnc-Query M, ID_R

1. $K \xleftarrow{\$} \{0,1\}^k$; $w = IV_2$;
2. $\mathsf{checkin}(M, r, k, \ell_{sg}) = m_1|| \dots ||m_{(n+1)}$
3. $C^*||T^* \xleftarrow{\$} \{0,1\}^{((n+1)*r)+k}$
4. $c_1^*||c_2^*|| \dots ||c_{n+1}^* = C^*$
5. $S^1||S^2||T = c_1^*|| \dots ||c_e^*||c_{e+1}^*|| \dots ||c_{n+1}^*||T^*$
6. $i \xleftarrow{\$} \{1 \dots q_H\}/I; S^2 = S_i;\ I = I \bigcup i$
7. $c_1^*|| \dots ||c_e^*||c_{e+1}^*|| \dots ||c_{n+1}^* = S^1||S^2$
8. $x = ID_S^*||ID_R||K$
9. **for** $i = 1 \to n+1$ **do**
   - $v = x||w;$
   - $((x = c_i^* \oplus m_i)||w) = \pi(x||w);$
   - $v' = x||w;\ I_\pi^{sc} = I_\pi^{sc} \bigcup \{v, v'\}$
   - $x = x \oplus m_i$
   - $c_i = x$
10. $(x||w) = \pi(x||w)$; $T = \lfloor x \rfloor_k$
11. $S^1||S^2||T = c_1^*|| \dots ||c_e^*||c_{e+1}^*|| \dots ||c_{n+1}^*||T^*$
12. $Y_1 = \mathsf{Enc}_{\mathsf{pk}_R}(S^1)$; $\sigma = \mathsf{Sign}_{\mathsf{sk}_S^*}(S^2)$; $\boxed{Y_1 = \mathsf{Enc}_{\mathsf{pk}_R}(S^1); \sigma = \sigma_i;}$
13. $pad(S^1||Y_1) = y_1|| \dots ||y_j$; $x = IV_1; w = IV_3$
14. **for** $i = 1 \to j$ **do**
   - $(x||w) = \pi(x \oplus y_i||w)$
15. $K_h = \lfloor x \rfloor_k \oplus K$; $T_k = T \oplus K$
16. Return: $(K_h, Y_1, Y_2 = (S^2, \sigma), T_k)$

Rest of oracles are same as G5

Figure 7.8: Game G8 and G9: Dash box shows dummy lines of codes added in G8 compare to G7. G8 follows code without oval-box. G9 follows code with Oval-box.

Now onward we start making changes in VerDec oracle.

Game 10: In Game **10** we add some dummy lines which doesn't affect UF-CMA experiment of Game and G**10** remain same as **G9**. In Game **10**, we modify VerDec oracle such that, we detect Existential forgery on VerDec and show a reduction to universal forgery on Ver. Whenever we discuss a forgery we consider $ID_R = ID_R^*$ in VerDec and related pk_{R^*} and sk_{R^*} for target sender ID_{S^*}.

We set a boolean value *flag* to *old* initially, and set it to *new* in case input-output response of π during VerDec not *belong* to (I_{sc}^{π} and $I_{\pi}^{\mathcal{A}}$) .

Here, $flag$ is *old* signifies that input to VerDec is output of $\mathsf{SignEnc}$ for some i^{th} query in case of $I_{vd} \subset I_{sc}^{\pi}$ or all π's input-output response already known to adversary $\mathcal{A}$ in $I_{\pi}^{\mathcal{A}}$ if $I_{vd} \subset I_{\pi}^{\mathcal{A}}$.

Similarly, if *flag* becomes *new* then one of the value of π in VerDec is new w.r.t $\mathsf{SignEnc}$. In case validation passed for *flag==new* then essentially answered M is not queried before to $\mathsf{SignEnc}$ and one of the values from K_h, Y_1, Y_2, T is differently used compare to any values in output of $\mathsf{SignEnc}$.

A forgery assumed to be valid only when $\mathsf{Ver}_{\mathsf{pk}_S}(y_2 = (S^2, \sigma)) \neq \perp$ and $T == T'$ happens under *flag==new* for ID_R^*. We try to detect a forgery based on chosen at random known input of Ver.

Game 11: In Game **G11**, we return $\perp$ in case of *flag* is *new*. Here difference between G10 the G11 will be probability of $T == T'$ in case of $flag\ new$.

In case validation passed for *flag=new* then essentially answered M is not queried before to $\mathsf{SignEnc}$ and one of the values from π is freshly defined. This leads to having target collision on propose T in input to VerDec. This happens with probability of $\frac{1}{2^k}$. Therefore, $|\Pr[G_{11}] - \Pr[G_{10}]| \leq \frac{q_{usc}}{2^k}$

Game $\boxed{\text{G10}}$ $\boxed{\text{G11}}$: Initialize same as G4

On SignEnc-Query M, ID_R

1. $K \xleftarrow{\$} \{0,1\}^k$; $w = IV_2$;
2. $\mathsf{checkin}(M, r, k, \ell_{sg}) = m_1 || \dots || m_{(n+1)}$
3. $C^* || T^* \xleftarrow{\$} \{0,1\}^{((n+1)*r)+k}$
4. $c_1^* || c_2^* || \dots || c_{n+1}^* = C^*$
5. $S^1 || S^2 || T =$
   $c_1^* || \dots || c_e^* || c_{e+1}^* || \dots || c_{n+1}^* || T^*$
6. $i \xleftarrow{\$} \{1 \dots q_H\}/I$; $S^2 = S_i$; $I = I \bigcup i$
7. $c_1^* || \dots || c_e^* || c_{e+1}^* || \dots || c_{n+1}^* = S^1 || S^2$
8. $x = ID_S^* || ID_R || K$
9. **for** $i = 1 \to n+1$ **do**
   - $v = x || w$;
   - $((x = c_i^* \oplus m_i) || w) = \pi(x || w)$;
   - $v' = x || w$; $I_\pi^{sc} = I_\pi^{sc} \bigcup \{v, v'\}$
   - $x = x \oplus m_i$
   - $c_i = x$
10. $(x || w) = \pi(x || w)$; $T = \lfloor x \rfloor_k$
11. $S^1 || S^2 || T =$
    $c_1^* || \dots || c_e^* || c_{e+1}^* || \dots || c_{n+1}^* || T^*$
12. $Y_1 = \mathsf{Enc}_{\mathsf{pk}_R}(S^1)$; $\sigma = \sigma_i$;
13. $pad(S^1 || Y_1) = y_1 || \dots || y_j$;
    $x = IV_1; w = IV_3$
14. **for** $i = 1 \to j$ **do**
    - $(x || w) = \pi(x \oplus y_i || w)$
15. $K_h = \lfloor x \rfloor_k \oplus K$; $T_k = T \oplus K$
16. Return: $(K_h, Y_1, Y_2 = (S^2, \sigma), T_k)$

On VerDec-Query (K_h, Y_1, Y_2, T_k)

1. $S^1 = \mathsf{Dec}_{\mathsf{sk}_R}(Y_1)$; $x = IV_1$; $w = IV_3$;
2. **if** $\mathit{Ver}_{\mathsf{pk}_{S^*}}(Y_2 = (S^2, \sigma)) == \perp$ **then**
   - Return $\perp$
3. $c_1 || \dots || c_e || c_{e+1} || \dots || c_{n+1} = S^1 || S^2$
4. $pad(S^1 || Y_1) = y_1 || \dots || y_j$; $\boxed{I_{vd} = \emptyset}$
5. **for** $i = 1 \to j$ **do**
   - $v = x || w$; $(x || w) = \pi(x || w)$;
   - $v' = x || w$; $\boxed{I_{vd} = \{v, v'\} \bigcup I_{vd}}$
6. $K = \lfloor x \rfloor_k \oplus K_h$; $T = T_k \oplus K$
7. $x = ID_S^* || ID_R || K$; $w = IV_2$;
   $\boxed{flag \leftarrow old}$
8. **for** $i = 1 \to (n+1)$ **do**
   - $v = x || w$; $(x || w) = \pi(x || w)$;
   - $v' = x || w$; $m_i = x \oplus c_i$; $x = c_i$;
   - $\boxed{I_{vd} = \{v, v'\} \bigcup I_{vd}}$
9. $v = x || w$; $(x || w) = \pi(x || w)$;
10. $v' = x || w$; $T' = \lfloor x \rfloor_r$;
    $\boxed{I_{vd} = \{v, v'\} \bigcup I_{vd}}$
11. $X' = m_1 || \dots || m_{n+1}$;
12. $\boxed{\textbf{if } I_{vd} \not\subset I_{sc} \;\&\&\; I_{vd} \not\subset I_\pi^{\mathcal{A}} \textbf{ then } flag \leftarrow new}$
13. **if** $T == T'$ $\boxed{and\ flag == new}$ **then**
    - **if** $\exists\ M$ *s.t.*
      $M = \mathsf{checkout}(X', r, k, \ell_{sg})$ **then**
      - $\boxed{\text{Return:} M}$ $\boxed{\text{Return: } \perp}$
    - **else**
      - Return: $\perp$

    else
    - $\perp$
14. **if** $T == T'$ $\boxed{and\ flag == old}$ **then**
    - **if** $\exists\ M$ *s.t.*
      $M = \mathsf{checkout}(X', r, k, \ell_{sg})$ **then**
      - Return: M
    - **else**
      - Return: $\perp$

    else
    - $\perp$

Rest same as G5

Figure 7.9: Game G10 and G11: Dash-box shows dummy line of code added in G10 compare to G9. G10 follows the code with oval-box without solid-box. G11 follows the code without oval-box with solid-box.

Game G12 G13: Initialize Same as G4 with Choose a $S_j \xleftarrow{\$} \{0,1\}^{\ell_{sg}}$

On SignEnc-Query M, ID_R

1. $K \xleftarrow{\$} \{0,1\}^k$; $w = IV_2$;
2. $\mathsf{checkin}(M, r, k, \ell_{sg}) = m_1||\ldots||m_{(n+1)}$
3. $C^*||T^* \xleftarrow{\$} \{0,1\}^{((n+1)*r)+k}$
4. $c_1^*||c_2^*||\ldots||c_{n+1}^* = C^*$
5. $S^1||S^2||T = c_1^*||\ldots||c_e^*||c_{e+1}^*||\ldots||c_{n+1}^*||T^*$
6. $i \xleftarrow{\$} \{1\ldots q_H\}/I; S^2 = S_i;\ I = I \bigcup i$
7. $c_1^*||\ldots||c_e^*||c_{e+1}^*||\ldots||c_{n+1}^* = S^1||S^2$
8. $x = ID_S^*||ID_R||K$
9. **for** $i = 1 \to n+1$ **do**
   - $v = x||w$;
   - $((x = c_i^* \oplus m_i)||w) = \pi(x||w)$;
   - $v' = x||w$; $I_\pi^{sc} = I_\pi^{sc} \bigcup \{v, v'\}$
   - $x = x \oplus m_i$
   - $c_i = x$
10. $(x||w) = \pi(x||w)$; $T = \lfloor x \rfloor_k$
11. $S^1||S^2||T = c_1^*||\ldots||c_e^*||c_{e+1}^*||\ldots||c_{n+1}^*||T^*$
12. **If** $S_i == S_j$ **then** Abort
13. $Y_1 = \mathsf{Enc}_{\mathsf{pk}_R}(S^1); \sigma = \sigma_i$;
14. $pad(S^1||Y_1) = y_1||\ldots||y_j$; $x = IV_1; w = IV_3$
15. **for** $i = 1 \to j$ **do**
   - $(x||w) = \pi(x \oplus y_i||w)$
16. $K_h = \lfloor x \rfloor_k \oplus K$; $T_k = T \oplus K$
17. Return: $(K_h, Y_1, Y_2 = (S^2, \sigma), T_k)$

On VerDec-Query (K_h, Y_1, Y_2, T_k)

1. $S^1 = \mathsf{Dec}_{\mathsf{sk}_R}(Y_1)$; $x = IV_1; w = IV_3$
2. **if** $\mathit{Ver}_{\mathsf{pk}_{S^*}}(Y_2 = (S^2, \sigma)) == \perp$ **then**
   - Return $\perp$
3. **if** $S^2 = S_j$ *and* $ID_R = ID_R^*$ **then**
   - $Bad_{sign} \leftarrow true$
4. $c_1||\ldots||c_e||c_{e+1}||\ldots||c_{n+1} = S^1||S^2$
5. $pad(S^1||Y_1) = y_1||\ldots||y_j$; $I_{vd} = \emptyset$; $flag \leftarrow old$
6. **for** $i = 1 \to j$ **do**
   - $(x||w) = \pi(x \oplus y_i||w)$
7. $K = \lfloor x \rfloor_k \oplus K_h$; $T = T_k \oplus K$
8. $x = ID_S^*||ID_R||K$; $w = IV_2$;
9. **for** $i = 1 \to (n+1)$ **do**
   - $v = x||w$; $(x||w) = \pi(x||w)$;
   - $v' = x||w$; $m_i = x \oplus c_i$;; $x = c_i$;
   - $I_{vd} = \{v, v'\} \bigcup I_{vd}$
10. $v = x||w$; $(x||w) = \pi(x||w)$;
11. $v' = x||w$; $T' = \lfloor x \rfloor_r$; $I_{vd} = \{v, v'\} \bigcup I_{vd}$
12. $X' = m_1||\ldots||m_{n+1}$;
13. **if** $I_{vd} \not\subset I_{sc}$ && $I_{vd} \not\subset I_\pi^{\mathcal{A}}$ **then**
   - $flag \leftarrow new$
14. **if** $T == T'$ *and* $flag == old$ **then**
   - **if** $I_{vd} \subseteq I_\pi^{\mathcal{A}}$ **then**
     - **if** $\exists\ M$ *s.t.* $M = \mathsf{checkout}(X', r, k, \ell_{sg})$ **then**
       - $Bad \leftarrow true$
       - Return: M $\boxed{\perp}$
     - **else**
       - Return: $\perp$
   - **if** $\exists\ M$ *s.t.* $M = \mathsf{checkout}(X', r, k, \ell_{sg})$ **then**
     - Return: M
   - **else**
     - Return: $\perp$

   else
   - $\perp$

Rest of Oracles are same as G5

Figure 7.10: Game G12 and G13: Dash-box shows added dummy line of code in G12 compare to G11. G12 follows the code without solid-box with Oval-box. G13 follows the code with solid-box without Oval-box.

In Game 12: Game **G12** is same as Game **G11** except some dummy lines of code added, shown in dash boxes.

- Initially a random S_j is chosen of length ℓ_{sg}. In case, this S_j appears in SignEnc during answering a query, we abort the SignEnc from answering. probability of such happening is $\frac{q_H}{2^{\ell}_{sg}}$ and this event is not helpful in forgery because such query is not providing any information to $\mathcal{A}$.
- We also mark an dummy event Bad_{sign} as true if during VerDec query $S^2 == S_j$ and $\mathsf{Ver}_{\mathsf{pk}_{S^*}}(y_2) = \top$ for $ID_R == ID^*_R$. This event signifies that adversary has provided a valid signature on random chosen S^2 for a targeted ID of sender and receiver. Later we show, probability of such Bad_{sign} is true is equivalent to $Adv^{uUF-RMA}_{\text{SIGN},\mathcal{B}}(k)$.
- We also mark an event as $Bad \leftarrow true$ in case VerDec returns M if $I_{vd} \subseteq I^{\mathcal{A}}_{\pi}$ is true and *flag* is still old.

In Game G13, we return $\perp$ instead of M in case $Bad \leftarrow true$. We check the probability of this Bad event to be happened. This event can be possible in either of two cases. We denote first case as Bad_{π} and second case as Bad_{sign}, e.g, $Pr[Bad \leftarrow true] = Pr[Bad_{\pi} \leftarrow true] + Pr[Bad_{sign} \leftarrow true]$.

Probability of $Bad_{\pi} \leftarrow true$ is as follows. This is the case when adversary has generated a valid ciphertext using individual query to $\pi^{\mathcal{A}}$ and with help of known message-signature pair. $\mathcal{A}$ could use custom K, pk^*_R and values of m_i so that adaptive calls of $\pi^{\mathcal{A}}$ will produce desired S^2 with known σ and some random T. Here, comes the part of special addition 0^r string block in message padding during checkin and checkout. This block force the $\mathcal{A}$ to select a particular K and values of m_i such that after producing S^1 next output of $\pi^{\mathcal{A}}$ should be equivalent to first r-bit of S^2. This is essential to pass the checkout function. Probability of such happening is $\frac{q_{sc}}{2^r}$ for available q_{sc} number of message-signature pairs through SignEnc queries.

Probability of $Bad_{sign} \leftarrow true$ is as follows. This case happens when adversary has generated a valid ciphertext using $I^{\mathcal{A}}_{\pi}$ having known individual query to $\pi^{\mathcal{A}}$ and *without* help of known message-signature pair by generating a valid signature for random S^2. This could happen as follows, adversary $\mathcal{A}$ ask queries to π for some random K,pk^*_R and custom m_i's in accordance to checkout function to generate random K_h, S^1, S^2 and T_k which will also validate $T == T'$ upon verification. Now in order to pass validation of Sign, $\mathcal{A}$ needs to have valid signature over Sign for random S^2. Because $\mathcal{A}$ knows the targeted message before generating signature, this becomes equivalent to universal forgery for random message. Therefore, $\Pr[Bad_{sign}] \leq Adv^{\mathsf{uUF\text{-}RMA}}_{\textsc{Sign}}(k)$

Therefore,adversary needs to produce either a collision over $r-$bit of S^2 using π query and known (S^2, σ) or in alternate way produce a valid signature over random S^2 which is output of π queries. Therefore, if adversary pass the checkout function then essentially $\mathcal{A}$ produces the collision. Probability of happen such collision is $\frac{q_{sc}}{2^r} + \Pr[Bad_{sign}]$. Therefore, $|\Pr[G_{13}] - \Pr[G_{12}]| \leq \frac{q_{sc}}{2^r} + Adv^{\mathsf{uUF\text{-}RMA}}_{\textsc{Sign}}(k)$
Game 14 and 13: Game **G14** is same as Game **G13**. **G14** is final ideal game and we simplify the cases by merging Bad event with $flag \leftarrow\ new$ event, because in both events VerDec is returning $\perp$. Now *flag* is set to new in case $(I_{vd} \not\subset I_{sc}$ && $I_{vd} \not\subset I^{\mathcal{A}}_{\pi})$ *or* $I_{vd} \subseteq I^{\mathcal{A}}_{\pi}$ and VerDec return $\perp$ if *flag* is new. Return of M will happen only if $flag$ is old and validation of T passed. Now essentially, $\mathcal{A}$ will get return $\perp$ for all his queries except either he produces a valid signature on any random S^2 not queried before or query the output of SignEnc.

This ends the proof.

□

We can have following corollaries from proof of lemma 1, which are also summarized in Table 7.2.

Corollary 1. *If the encryption scheme follows OW-PCA, and the signature scheme is* uUF-RMA*, then the parallel signcryption scheme is* UF-AdA*.*

Corollary 1 is direct implication from Lemma 1. This corollary includes both probabilistic and deterministic signature schemes and also encryption schemes.

Corollary 2 is a sub-class result from Corollary 1, where deterministic signature scheme follows UF-AdA(or signature schemes follow sUF-AdA). This corollary serves as a bridge for our next corollary 3.

Corollary 2. *If the encryption scheme follows OW-PCA, and the signature scheme is* suUF-RMA*, then the parallel signcryption scheme is* UF-AdA*.*

Corollary 3. *If the encryption scheme is deterministic and follows one-wayness, and the signature scheme is* suUF-RMA*, then the parallel signcryption scheme is* sUF-AdA*.*

Corollary 2 and Corollary 3 has a difference in achieved security because of probabilistic and deterministic nature of encryption scheme. This is mainly because encryption scheme which follows OW-PCA includes some probabilistic asymmetric encryption schemes which have re-randomization problem. In re-randomization, for same input to asymmetric primitive a different value output could be generated. In such a case and because of insider security model, adversary attacking unforgeability of SIGNCRYPT can produce a different sign-ciphertext for same input message which is quired earlier. For example, for a query M, ID_R, output is K_h, Y_1, Y_2, T for a K. Using insider knowledge and probabilistic nature of asymmetric encryption new valid output could be K'_h, Y'_1, Y_2, T for same K and M, ID_R. Such a valid pair is allowed as part of forgery in sUF, but not in UF. Therefore, in corollary 2, Sign follows suUF-RMA, but overall SIGNCRYPT follows only UF-AdA. If encryption scheme is deterministic then above attack not valid and SIGNCRYPT can be benefited from suUF-RMA. A summary of above discussed corollary is shown in table 7.2.

Encrypt(↓) \ Sign(→)		uUF-RMA	suUF-RMA
Deterministic	OW-CPA	UF-AdA	sUF-AdA
Probabilistic	OW-PCA	UF-AdA	UF-AdA

Table 7.2: Unforgeability of SIGNCRYPT in different assumption on Sign and Encrypt.

Game G14: Initialize $I_\pi^{sc} = I = I_\pi = I_\pi^{\mathcal{A}} = \emptyset$, $IV_1 = 0^r$ $IV_2 = 0^c$, $(\mathsf{sk}_R, \mathsf{pk}_R) \leftarrow \mathsf{GenEnc}(1^k)$, pk_S^*, ID_S^*;
`Signlist` : $\{(S_i, \sigma_i) : \sigma_i = \mathsf{Sign}_{\mathsf{sk}_S}(S_i)\ \forall 1 \le i \le q_H$ and each S_i chosen randomly$\}$.Choose a $S_j \xleftarrow{\$} \{0,1\}^{\ell_{sg}}$

On SignEnc-Query M, ID_R

1. $K \xleftarrow{\$} \{0,1\}^k$; $w = IV_2$;
2. $\mathsf{checkin}(M, r, k, \ell_{sg}) = m_1 || \ldots || m_{(n+1)}$
3. $C^* || T^* \xleftarrow{\$} \{0,1\}^{((n+1)*r)+k}$
4. $c_1^* || c_2^* || \ldots || c_{n+1}^* = C^*$
5. $S^1 || S^2 || T =$ $c_1^* || \ldots || c_e^* || c_{e+1}^* || \ldots || c_{n+1}^* || T^*$
6. $i \xleftarrow{\$} \{1 \ldots q_H\}/I$;$S^2 = S_i$; $I = I \bigcup i$
7. $c_1^* || \ldots || c_e^* || c_{e+1}^* || \ldots || c_{n+1}^* = S^1 || S^2$
8. $x = ID_S^* || ID_R || K$
9. **for** $i = 1 \to n+1$ **do**
   - $v = x||w$;
   - $((x = c_i^* \oplus m_i)||w) = \pi(x||w)$;
   - $v' = x||w$; $I_\pi^{sc} = I_\pi^{sc} \bigcup \{v, v'\}$
   - $x = x \oplus m_i$
   - $c_i = x$
10. $(x||w) = \pi(x||w)$; $T = \lfloor x \rfloor_k$
11. $S^1 || S^2 || T =$ $c_1^* || \ldots || c_e^* || c_{e+1}^* || \ldots || c_{n+1}^* || T^*$
12. **If** $S_i == S_j$ **then** Abort
13. $Y_1 = \mathsf{Enc}_{\mathsf{pk}_R}(S^1)$; $\sigma = \sigma_i$;
14. $pad(S^1||Y_1) = y_1 || \ldots || y_j$; $x = IV_1; w = IV_3$
15. **for** $i = 1 \to j$ **do**
    - $(x||w) = \pi(x \oplus y_i || w)$
16. $K_h = \lfloor x \rfloor_k \oplus K$
17. Return: $(K_h, Y_1, Y_2 = (S^2, \sigma), T)$

On VerDec-Query (K_h, Y_1, Y_2, T)

1. $S^1 = \mathsf{Dec}_{\mathsf{sk}_R}(Y_1)$; $x = IV_1; w = IV_3$;
2. **if** $Ver_{pk_S}(Y_2 = (S^2, \sigma)) == \perp$ **then**
   - Return $\perp$
3. **if** $S^2 == S_j$ *and* $ID_R == ID_R^*$ **then**
   - $Bad_{sign} \leftarrow true$; $\sigma^* = \sigma$
4. $c_1 || \ldots || c_e || c_{e+1} || \ldots || c_{n+1} = S^1 || S^2$
5. $pad(S^1||Y_1) = y_1 || \ldots || y_j$;
6. **for** $i = 1 \to j$ **do**
   - $(x||w) = \pi(x \oplus y_i || w)$
7. $K = \lfloor x \rfloor_k \oplus K_h$ $I_{vd} = \emptyset$
8. $x = ID_S^* || ID_R || K$; $flag \leftarrow old$
9. **for** $i = 1 \to (n+1)$ **do**
   - $v = x||w$;
   - $(x||w) = \pi(x||w)$
   - $v' = x||w$;
   - $m_i = x \oplus c_i$
   - $x = c_i$
   - $I_{vd} = \{v, v'\} \bigcup I_{vd}$
10. $v = x||w$; $(x||w) = \pi(x||w)$;
11. $v' = x||w$; $T' = \lfloor x \rfloor_r$; $I_{vd} = \{v, v'\} \bigcup I_{vd}$
12. $X' = m_1 || \ldots || m_{n+1}$;
13. **if** $(I_{vd} \not\subset I_{sc}\ \&\&\ I_{vd} \not\subset I_\pi^{\mathcal{A}})$ *or* $I_{vd} \subseteq I_\pi^{\mathcal{A}}$ **then**
    - $flag \leftarrow new$
14. **if** $T == T'$ *and* $flag == old$ **then**
    - **if** $\exists\ M$ *s.t.* $M = \mathsf{checkout}(X', r, k, \ell_{sg})$ **then**
      - Return:M
    - **else**
      - Return: $\perp$

    else
    - $\perp$

Rest of Oracles are same as G5

Figure 7.11: Game G14: G14 same as G13 with simplified code.

Adversary $\mathcal{B}$ Initialize: Given a ID_S^* and ID_R^*. public/pvt key pair of R^* as pk_{R^*} and sk_{R^*}. Public key of S^* as pk_{S^*} and a Target message M^*. We also denote M^* as μ or S_j.

$\mathcal{A}$: Initialize $I_\pi^{sc} = I = I_\pi = I_\pi^{\mathcal{A}} = \emptyset$, $IV_1 = 0^r$ $IV_2 = 0^c$, $(\mathsf{sk}_R,\mathsf{pk}_R) \leftarrow \mathsf{GenEnc}(1^k)$, pk_S^*, ID_S^*;

`Signlist` : $\{(S_i, \sigma_i) : \sigma_i = \mathsf{Sign}_{\mathsf{sk}_S}(S_i)\ \forall 1 \le i \le q_H$ and each S_i chosen randomly$\}$.

Rest of Oracles are same as G14

Finalization: If VerDec return M and $Bad_{sign} \leftarrow true$ then return σ^*.

Figure 7.12: Adversary $\mathcal{B}$ over uUF-RMA

Indistinguishability

Proof Sketch: We are dealing with insider security model in the multi-user setting, the adversary has a target receiver ID_R^* in mind. The adversary knows the receiver public key pk_R and has access to the VerDec oracle under sk_R. Further, we assume that an adversary $\mathcal{A}$ observed q_{usc} queries to VerDec oracle. $\mathcal{A}_1$ has also chosen a pair of messages M_0 and M_1 and a key pair (sk_S^*, pk_S^*) for ID_S^*. It receives a ciphertext (y_1^*, y_2^*, y_3^*) under (sk_S^*, pk_R^*) of either M_0 or M_1. The unknown message is denoted by M_d, where d is the bit that adversary $\mathcal{A}_2$ wishes to find out.

Indistinguishability proof of SIGNCRYPT follows the security proof of SpPad–Pewithout much trouble and difference. Because of insider security model adversary knows input of Sign, which is also known in SpPad–Pe conceptually.

We make subsequent changes in permutation π such that π gives a permutation response for each new query but r bits out of b-bit output are random. Likewise, c bits out of b-bit output are always different for new input. This part remains same as for unforgeability.

Modify unsigncryption oracle such that it nullifies those queries to unsigncryption oracle about which adversary does not know answer in advance with help of π query and which can be simulated without using private key of receiver sk_{R^*}. If $I_\pi^{vd} \not\subset I_\pi^{\mathcal{A}}$, then probability that adversary can get an answer from unsigncryption oracle is bounded by $\frac{q_{usc}}{2^k}$ which includes target collision on T for q_{usc} number of unsigncryption queries. Unlike unforgeability, adversary is allowed to generate valid signcryptext but only those will be valid about which adversary already knows the answer.

Modifying signcryption oracle using random response of π. This will lead to simulate signcryption oracle to return a random response. This change will be bounded by the probability of guessing the used randomness K by adversary or advantage of an OW-PCA adversary to break the one-wayness (OW).

Privacy proof of scheme depends upon probabilistic or deterministic nature of underlying signature scheme. During the proof we assume signature scheme is deterministic and follows the correctness condition. In subsequent section we show how we can remove this assumption on signature scheme.

Following lemma can be derived from the Theorem 8:

Lemma 2. *Consider an adversary $\mathcal{A}$ against the IND-CCA security of the parallel signcryption scheme with advantage $Adv^{IND-CCA}_{Signcrypt}(k)$ whose running time is bounded by t and which makes at most q_π queries to permutation $\pi : \{0,1\}^{b=r+c} \rightarrow \{0,1\}^b$ oracle and q_{usc} queries to the un-signcryption oracle. Then there exists an attacker $\mathcal{B}$ against the OW-PCA security of the public key encryption scheme with advantage $Adv^{OW-PCA}_{\text{ENCRYPT}}(k)$ and whose running time bounded by $t' \leq t + q_{usc}(\tau + O(1))$, where τ denotes the maximal running time of decryption and verification algorithms, for which*
$Adv^{IND-CCA}_{Signcrypt}(k) \leq Adv^{OW-PCA}_{\text{ENCRYPT}}(k) + \frac{(q_\pi-1)q_\pi}{2^{b+1}} + \frac{q_\pi(q_\pi+1)}{2^c} + \frac{q_{usc}}{2^k} + \frac{q_\pi^{\mathcal{A}}}{2^k}$
, where q_π is total number of π queries including queries by adversary ($q_\pi^{\mathcal{A}}$), signcryption and unsigncryption oracle.

Proof. (of Lemma 2) We consider the following experiment for IND-CCA experiment for SIGNCRYPT by Adversary $\mathcal{A}$:

Experiment: $Exp^{ind-cca-d}_{\mathsf{SIGNCRYPT},\mathcal{A}}(k)$:

1. $(\mathsf{sk}_{R^*},\mathsf{pk}_{R^*}) \leftarrow \mathsf{GenEnc}_{R^*}(1^k)$
2. $(M_0, M_1, ID_{S^*}) \leftarrow \mathcal{A}^{\mathsf{VerDec}_{\mathsf{sk}_{R^*}}(\cdot,\cdot),\pi(\cdot)}(\mathsf{pk}_{R^*})$
3. Mapping to pk_{S^*} using ID_{S^*}, where $(\mathsf{sk}_{S^*},\mathsf{pk}_{S^*}) \leftarrow \mathsf{GenSig}_S(1^k)$
4. $d \xleftarrow{\$} \{0,1\}$
5. $y^* \leftarrow \mathsf{SignEnc}_{\mathsf{pk}_{R^*},\mathsf{sk}_{S^*}}(M_d)$
6. $d' \leftarrow \mathcal{A}^{\mathsf{VerDec}_{\mathsf{sk}_{R^*}}(\cdot,\cdot),\pi(\cdot)}(\mathsf{pk}_{R^*}, \mathsf{sk}_{S^*}, \mathsf{pk}_{S^*}, y^*)$
7. **if** $d == d'$ *and* (y^*, ID_{S^*}) *query never made to* $\mathit{VerDec}_{\mathsf{sk}_{R^*}}(\cdot,\cdot)$ *oracle* **then**
   Return 1

   else

   Return 0

Advantage of adversary $\mathcal{A}$ is given by following probability:

$$Adv_{Signcrypt}^{IND-CCA}(k) = \mid \Pr[Exp_{\mathsf{SIGNCRYPT},\mathcal{A}}^{ind-cca-d}(k) = 1 | d \xleftarrow{\$} \{0,1\}] - \tfrac{1}{2} \mid .$$

We are dealing with insider security model in the multi-user setting, the adversary has a target receiver ID_R^* in mind. The adversary knows the receiver public key pk_R and has access to the VerDec oracle under sk_R. Further, we assume that an adversary $\mathcal{A}$ observed q_{usc} queries to VerDec oracle. $\mathcal{A}$ also chosen a pair of messages M_0 and M_1 and a key pair (sk_S, pk_S) for ID_S. It receives a ciphertext (y_1^*, y_2^*, y_3^*) under (sk_S, pk_R) of either M_0 or M_1. The unknown message is denoted by M_d, where d is the bit the adversary wishes to find out. The adversary $\mathcal{A}$ output a bit d' which is equal to d with advantage ϵ,i.e., $\Pr[d' = d]$=$1/2 + \epsilon$. In the following, we use a * for all internal values used in computing the challenge signcryption.

We will use game playing techniques [15,16]. We start from original CCA game $EXP_{\mathsf{SIGNCRYPT},\mathcal{A}}$ or $Exp_{\mathsf{SIGNCRYPT},\mathcal{A}}^{ind-cca-d}(k) = 1 | d \xleftarrow{\$} \{0,1\}$ denote the event that $\mathcal{A}$ outputs $d' = d$ where $d \xleftarrow{\$} \{0,1\}$. We want to show that $|\Pr[EXP_{\mathsf{SIGNCRYPT},\mathcal{A}}]| = \frac{1}{2} + negl(k)$, where $negl(\cdot)$ is a negligible function and $negl(k) \leq Adv_{Signcrypt}^{IND-CCA}(k)$. In each game, following set maintained: I by π, $I_\pi^{\mathcal{A}}$ by $\pi_{\mathcal{A}}$ and Y stores capacity c-bit values upon each query to π.

We modify $Signcrypt$ into a sequence of game $G0, G1, \ldots G12$ such that:

$Pr[EX_{\mathcal{F}-SpAEP,\mathcal{A}}]$= $Pr[EX_{G0,\mathcal{A}}]$
$Pr[EX_{G(i-1),\mathcal{A}}]$=$Pr[EX_{Gi,\mathcal{A}}]$+$negl(k)$ $\forall 1 \leq i \leq 11$
$Pr[EX_{G12,\mathcal{A}}]$= $\frac{1}{2}$

Game G0 to G5: From Game G0 to G5 it follows changes exactly same as in Proof of Lemma 1. Therefore, $|\Pr[EXP_{G0,\mathcal{A}}] - \Pr[EXP_{G5,\mathcal{A}}]| \leq \frac{(q_\pi - 1)q_\pi}{2^{b+1}} + \frac{q_\pi(q_\pi+1)}{2^c}$.

In G5, game maintains an extra set I_{enc} which stores input-output response of π (as π_{enc}) during SignEnc challenge query.

Game G5 Initialize $I_{enc} = I_\pi = I_\pi^{\mathcal{A}} = \emptyset$, $IV_1 = 0^r$ $IV_2 = 0^c$, $IV_3 = IV_2 \oplus 1$, $\mathcal{L}_c = \{IV_2, IV_3\}$, $(\mathsf{sk}_S, \mathsf{pk}_S) \leftarrow \mathsf{GenEnc}(1^k)$, pk_R^*, ID_R^*;

On SignEnc-Query M_d for ID_S^*

1. $K^* \xleftarrow{\$} \{0,1\}^k$;
   $x = IV_1 = ID_S^* || ID_R^* || 0^k$; $w = IV_2$;
2. $\mathsf{checkin}(M, r, k, \ell_{sg}) = m_1 || \dots || m_{(n+1)}$
3. $x = ID_S^* || ID_R^* || K$
4. **for** $i = 1 \to (n+1)$ **do**
   - $(x||w) = \pi_{enc}(x||w)$
   - $x = x \oplus m_i$
   - $c_i^* = x$
5. $(x||w) = \pi_{enc}(x||w)$; $T^* = \lfloor x \rfloor_k$
6. $S^{1*} || S^{2*} || T^* =$
   $c_1^* || \dots || c_e^* || c_{e+1}^* || \dots || c_{n+1}^* || T^*$
7. $Y_1^* = \mathsf{Enc}_{\mathsf{pk}_R^*}(S^1)$; $\sigma^* = \mathsf{Sign}_{\mathsf{sk}_S^*}(S^{2*})$;
8. $pad(S^{1*} || Y_1^*) = y_1^* || \dots || y_j^*$;
   $x = IV_1; w = IV_3$
9. **for** $i = 1 \to j$ **do**
   - $(x||w) = \pi_{enc}(x \oplus y_i^* || w)$
10. $K_h^* = \lfloor x \rfloor_k \oplus K^*$; $T_k^* = T^* \oplus K^*$
11. Return: $(K_h^*, Y_1^*, Y_2^* = (S^{2*}, \sigma^*), T_k^*)$

On VerDec-Query (K_h, Y_1, Y_2, T_k)

1. $S^1 = \mathsf{Dec}_{\mathsf{sk}_{R^*}}(y_1)$; $x = IV_1$; $w = IV_3$;
2. **if** $\mathit{Ver}_{\mathsf{pk}_S}(Y_2 = (S^2, \sigma)) == \perp$ **then**
   - Return $\perp$
3. $c_1 || \dots || c_e || c_{e+1} || \dots || c_{n+1} = S^1 || S^2$
4. $y_1 || \dots || y_j = pad(S^1 || Y_1)$;
5. **for** $i = 1 \to j$ **do**
   - $(x||w) = \pi(x \oplus y_i || w)$
6. $K = \lfloor x \rfloor_k \oplus K_h$; $T = T_k \oplus K$
7. $x = ID_S || ID_R^* || K$; $w = IV_2$;
8. **for** $i = 1 \to n+1$ **do**
   - $(x||w) = \pi(x||w)$
   - $m_i = x \oplus c_i$
   - $x = c_i$
9. $(x||w) = \pi(x||w)$; $T' = \lfloor x \rfloor_k$
10. $X' = m_1 || \dots || m_{n+1}$;
11. **if** $T == T'$ **then**
    - **if** $\exists\ M$ $s.t.$
      $M = \mathsf{checkout}(X', r, k, \ell_{sg})$ **then**
      - Return:M
    - **else**
      - Return: $\perp$

    else
    - $\perp$

On π-Query m, where $m \in \{0,1\}^b$

1. let $(x||w) = m$, where $x \in \{0,1\}^r$,
   $w \in \{0,1\}^c$,
2. if $(m, v) \in I_\pi$ then return v
3. $v_1 || v_2 \xleftarrow{\$} \{0,1\}^b$, where $v_1 \in \{0,1\}^r$,
   $v_2 \in \{0,1\}^c$,
4. if $v_2 \in Y \bigcup \{w\}$, then
   $v_2 \xleftarrow{\$} \{0,1\}^c \setminus Y \bigcup \{w\}$
5. $I_\pi = I_\pi \bigcup \{(m, v_1 || v_2)\}$ and
   $Y = Y \bigcup \{v_2, w\}$
6. return $v = v_1 || v_2$;

On π^{-1}-Query v. where $v \in \{0,1\}^b$

1. let $(v_1 || v_2) = v$, where
   $v_1 \in \{0,1\}^r$, $v_2 \in \{0,1\}^c$,
2. if $(m, v) \in I_\pi$ then return m
3. $m' || m'' \xleftarrow{\$} \{0,1\}^b$, where $m' \in \{0,1\}^r$,
   $m'' \in \{0,1\}^c$
4. if $m'' \in Y \bigcup \{v_2\}$, then
   $m'' \xleftarrow{\$} \{0,1\}^c \setminus Y \bigcup \{v_2\}$
5. $I_\pi = I_\pi \bigcup \{(m' || m'', v)\}$ and
   $Y = Y \bigcup \{m'', v_2\}$
6. return $m = m' || m''$;

On π_{enc}-Query m

1. $v = \pi(m)$
2. $I_{enc} = I_{enc} \bigcup (m, v)$

On $\pi_{\mathcal{A}}$-Query m

Same as G0

On $\pi_{\mathcal{A}}^{-1}$-Query v

Same as G0

Figure 7.13: Game G5

Game G6 and G5: Both the games are same. In Game G6, a dummy operation of $flag \leftarrow new$ is added in the VerDec oracle to denote a new query. The query is new in the sense that neither the query nor any part of the query during internal calls to π was queried earlier by the adversary. That is, $flag \leftarrow new$ if any π's response $\notin I_\pi^{\mathcal{A}}$. Now, code of Game G6 can check condition $T == T'$ in case of $flag == new$ and in case of $flag == old$ separately. If $T == T' \&\& \ flag == new$ then we mark this event as $\mathbf{bad}_\pi \leftarrow true$. Because event $\mathbf{bad}_\pi$ is just dummy event and return by VerDec in G6 is not affected therefore, $|\Pr[EXP_{G6,\mathcal{A}}]| = |\Pr[EXP_{G5,\mathcal{A}}]|$.

Game G7 and G6: In Game G7, in VerDec, we return $\perp$ instead of M in case $\mathbf{bad}_\pi$ is true. Therefore,

$|\Pr[EXP_{G7,\mathcal{A}}] - \Pr[EXP_{G6,\mathcal{A}}]| \leq \Pr[\mathbf{bad}_\pi \leftarrow true]$.

Let $(v_1||v_2) = \pi(x||w)$, where $x,v_1 \in \{0,1\}^r$ and $w,v_2 \in \{0,1\}^c$. In VerDec, a input is a *new query* to π when $((x||w),(v_1||v_2)) \notin I_\pi^{\mathcal{A}}$ and *old query* when $((x||w),(v_1||v_2)) \in I_\pi^{\mathcal{A}}$. If a *new query* $(x||w)$ is input to π during VerDec, then π outputs $v_1||v_2$, where $v_2 \notin \mathcal{L}_c$. That is, v_2 is also new. Since v_2 is unseen so far, it ensures that the input to the next call of π is certainly new. Further, since v_2 is new, next input $x'||v_2$ satisfies the condition $(x'||v_2, *) \notin I_\pi^{\mathcal{A}}$, where $*$ stands for any b bit value. Therefore one *new query* makes all subsequent inputs to $\pi(\cdot)$ as new. We already know for any new query r-bit response of π is random. Therefore in case of *flag* is *new* probability of $T == T'$ is equivalent to collision over k-bit T value. Therefore, $\Pr[\mathbf{bad}_\pi \leftarrow true] = \frac{q_{usc}}{2^k}$ for q_{usc} number of VerDec queries. Therefore,

$|\Pr[EXP_{G7,\mathcal{A}}] - \Pr[EXP_{G6,\mathcal{A}}]| \leq \frac{q_{usc}}{2^k}$.

Now, if this bad event does not happen then $G7$ will return M only in case all π response already known to $\mathcal{A}$. Consecutively, $\mathcal{A}$ already knows the answer of VerDec with help of π queries and available Sign, Ver and Enc functions.

Game $\boxed{\textbf{G6}}$ $\boxed{\textbf{G7}}$**:** Initialize $I_{enc} = I_\pi = I_\pi^{\mathcal{A}} = \emptyset$,$\mathsf{GenEnc}(1^k)$, pk_R^*, ID_R^*;, $IV_1 = 0^r, IV_2 = 0^c$, $IV_3 = IV_2 \oplus 1$. $\mathcal{L}_c = \{IV_2, IV_3\}$. $\boxed{flag \in \{new, old\}}$.

On Decryption-Query K_h, Y_1, Y_2, T_k

1 $S^1 = \mathsf{Dec}_{sk}(Y_1)$; $x = IV_1$ and $w = IV_3$

2 **if** $\mathit{Ver}_{\mathsf{pk}_S}(Y_2 = (S^2, \sigma)) == \perp$ **then**
  Return $\perp$

3 $c_1||\ldots||c_e||c_{e+1}||\ldots||c_{n+1} = S^1||S^2$;

4 $pad(S^1||Y_1)$=$y_1||\ldots||y_j$; $\boxed{flag \leftarrow old}$

5 **for** $i = 1 \to j$ **do**
  $x = x \oplus y_i$
  $\boxed{\textbf{If}\{x||w, *\} \notin I_\pi^{\mathcal{A}} \textbf{ then } flag \leftarrow new}$
  $(x||w) = \pi(x||w)$

6 $h = \lfloor x \rfloor_k$; $K = h \oplus K_h$; $T = T_k \oplus K$

7 $x = ID_S||ID_R||K$; $w = IV_2$;

8 **for** $i = 1 \to (n+1)$ **do**
  $\boxed{\textbf{If}\{x||w, *\} \notin I_\pi^{\mathcal{A}} \textbf{ then } flag \leftarrow new}$
  $(x||w) = \pi(x||w)$; $m_i = x \oplus c_i$; $x = c_i$

9 $\boxed{\textbf{If}\{x||w, *\} \notin I_\pi^{\mathcal{A}} \textbf{ then } flag \leftarrow new}$

10 $(x||w) = \pi(x||w)$; $T' = \lfloor x \rfloor_k$; $X' = m_1||\ldots||m_{n+1}$;

11 **if** $T = T'$ *and* $flag = new$ **then**
  $\boxed{\textbf{bad}_\pi \leftarrow true}$
  if $\exists\ M\ s.t.\ M = \mathsf{checkout}(X', r, k, \ell_{sg})$ **then**
    $\boxed{\text{Return:}M}$ $\boxed{\text{Return } \perp}$
  else
    Return: $\perp$

12 **if** $T = T'$ *and* $flag = old$ **then**
  if $\exists\ M\ s.t.\ M = \mathsf{checkout}(X', r, k, \ell_{sg})$ **then**
    Return:M
  else
    Return: $\perp$
else
  $\perp$

Rest of Oracles same as G5

Figure 7.14: Game G6: G6 includes dummy lines, shown in dash-box,compare to G5 along with round-box

Figure 7.15: G7: G7 includes all codes of line of G6 and also solid-box except round-box.

Game G8: Initialize $I_{enc} = I_\pi = I_\pi^{\mathcal{A}} = \emptyset$, $\mathsf{GenEnc}(1^k)$, pk_R^*, ID_R^*;, $IV_1 = 0^r, IV_2 = 0^c$, $IV_3 = IV_2 \oplus 1$. $\mathcal{L}_c = \{IV_2, IV_3\}$

On Decryption-Query K_h, Y_1, Y_2, T_k

1 **if** $\mathit{Ver}_{\mathsf{pk}_S}(Y_2 = (S^2, \sigma)) == \perp$ **then**
  Return $\perp$
2 If $\exists$ $\mathsf{checkin}(M, r, k, \ell_{sg}) = m_1||m_2||\ldots||m_{n+1}$ such that
after setting $Y_1 = a_{e+1}||\ldots||a_j$, $u_{2_1} = IV_3$, $z_{1_1} = IV_1$
if $\{(u_{1_i}||u_{2_i}), (z_{1_i+1}||z_{2_i+1})\} \in I_\pi^{\mathcal{A}}$ for $i : 1 \to e \to j$ such that $a_i = u_{1_i} \oplus z_{1_i}$, $u_{2_i} = z_{2_i}$ and $\mathcal{O}^{PC}(S^1, Y_1) = 1$, where $S^1 = a_1||\ldots||a_e$
then for setting $K = \lfloor z_j \rfloor_r \oplus K_h$, $S^1||S^2 = c_1||\ldots c_e||c_{e+1}||\ldots||c_n$, $x_0 = K||0^{r-k} \oplus IV_1$, $T = T_k \oplus K$ and $w_0 = IV_2$
if $(x_0||w_0, v_{1_1}||v_{2_1}) \in I_\pi^{\mathcal{A}}$ and
$\{(x_i||w_i), (v_{1_{i+1}}||v_{2_{i+1}})\} \in I_\pi^{\mathcal{A}}$ for $i : 1 \to e \to n+1$ and
$\lfloor v_{1_{n+2}} \rfloor_r == T$ where $x_i = c_i = m_i \oplus v_{1_i}$,$w_i = v_{2_i}$
then **return** M
else **Return** $\perp$
else **Return** $\perp$

Rest of Oracles same as G7

Following special notations is begin used during Game G8 and onwards in decryption oracle:

1. During SpongeWrap part of SpPad, we represent input-output relation of π's subsequent calls for $pad(M) = m_1||\ldots||m_n$ by $(v_{1_{i+1}}||v_{2_{i+1}}) = \pi(x_i||w_i)$, where $x_i = v_{1_i} \oplus \{m_i\}$, $w_i = v_{2_i}$ $0 \leq i \leq n$, $v_{1_0} = IV_1$,$m_0 = K$, $w_0 = IV_2$, $v_{1_i}, x_i \in \{0,1\}^r$ and $v_{2_i}, w_i \in \{0,1\}^c$. Then c_i will represent $m_i \oplus v_{1_i}$, here $1 \leq i \leq n$.

2. Input-output relation of π's subsequent call during *Sponge* part of SpPad will be represented as follows: $(z_{1_i+1}||z_{2_i+1}) = \pi(u_{1_i}||u_{2_i})$,$u_{1_i} = c_i \oplus z_{1_i}$, $u_{2_i} = z_{2_i}$,where $1 \leq i \leq (j)$, $u_{2_1} = IV_3$, $z_{1_1} = IV_1$, $z_j = h$.

Figure 7.16: Game G8: Output of decryption oracle in G8 is same as G7 but independent from sk.

Game G8: Both the games are same. Game G7 and G8 both return $\perp$ when a *new query* is given to the VerDec oracle. In Game G8, a message M is returned only when all the input-output relations of π, which would be possible during the encryption of M, are already in $I_\pi^{\mathcal{A}}$. Game G8 iterates over all the possible pairs of (input,output) of $\pi \in I_\pi^{\mathcal{A}}$ starting using IV_1 and IV_3 and tries to find a S^1 such that $\mathcal{O}^{PC}(S^1, Y^1) = 1$. In positive case, it further calculate K and then tries to find all pairs of input-output response which reaches to T via K, S^1, S^2. If any of response is missing then VerDec simply rejects the query. *Due to insider model, a faithful assumption on signing algorithm we have is for same input to signing algorithm two different signature can not be generated. We will discuss the impact of this assumption later, after the proof.*

Game G8 and Game G9: We start incremental changes in *Signcryption* oracle from Game G9. In Game G9, K^* is chosen before signcryption query and after "find" stage. In both case K^* remain random therefore,

$$|Pr[EXP_{G9,\mathcal{A}}] = Pr[EXP_{G8,\mathcal{A}}]|.$$

Some extra dummy variables are also chosen S^{1*}, S^{2*}, T^*, along with K^*, after find stage but not used. A dummy value Y_1^* is calculated on S^{1*} using Enc.

Game G9 $\boxed{\textbf{G10}}$**:**Initialize

$I_{enc} = I_\pi = I_\pi^{\mathcal{A}} = \emptyset$, $\mathsf{GenEnc}(1^k)$, pk_R^*, ID_R^*;, $IV_1 = 0^r, IV_2 = 0^c$, $IV_3 = IV_2 \oplus 1$. $\mathcal{L}_c = \{IV_2, IV_3\}$

After Find Stage(AFS): $K_h^* \xleftarrow{\$} \{0,1\}^k$; $S^{1*} \xleftarrow{\$} \{0,1\}^\ell$, $S^{2*} \xleftarrow{\$} \{0,1\}^{\ell_{sg}}$

$Y_1^* = \mathsf{Enc}(S^{1*})$, $T^* \xleftarrow{\$} \{0,1\}^k$

$y_1||\ldots||y_e||y_{e+1}||\ldots||y_j = S^{1*}||Y_1^*$;

$K^* \xleftarrow{\$} \{0,1\}^k$

$\boxed{K^*||* = \pi_{enc}(\ldots\pi_{enc}(\pi_{enc}(y_1 \oplus IV_1||IV_3) \oplus y_2||0^c)\ldots \oplus y_j||0^c) \oplus K_h^*||0^{b-k}}$

On Encryption-Query(M_d)

1. $K^* \xleftarrow{\$} \{0,1\}^k$;
   $x = IV_1 = ID_S^*||ID_R^*||0^k$;
   $w = IV_2$;
2. $\mathsf{checkin}(M, r, k, \ell_{sg}) = m_1||\ldots||m_{(n+1)}$
3. $x = ID_S^*||ID_R^*||K$
4. **for** $i = 1 \to (n+1)$ **do**
   $(x||w) = \pi_{enc}(x||w)$
   $x = x \oplus m_i$
   $c_i^* = x$
5. $(x||w) = \pi_{enc}(x||w)$;
   $T^* = \lfloor x \rfloor_k$
6. $S^{1*}||S^{2*}||T^* = c_1^*||\ldots||c_e^*||c_{e+1}^*||\ldots||c_{n+1}^*||T^*$
7. $Y_1^* = \mathsf{Enc}_{\mathsf{pk}_R^*}(S^1)$;
   $\sigma^* = \mathsf{Sign}_{\mathsf{sk}_S^*}(S^{2*})$;
8. $pad(S^{1*}||Y_1^*) = y_1^*||\ldots||y_j^*$;
   $x = IV_1; w = IV_3$
9. **for** $i = 1 \to j$ **do**
   $(x||w) = \pi_{enc}(x \oplus y_i^*||w)$
10. $K_h^* = \lfloor x \rfloor_k \oplus K^*$; $T_k^* = T \oplus K$
11. Return:
    $(K_h^*, Y_1^*, Y_2^* = (S^{2*}, \sigma^*), T_k^*)$

Rest of Oracles same as G8

G11: Initialize $I_{enc} = I_\pi = I_\pi^{\mathcal{A}} = \emptyset$, $\mathsf{GenEnc}(1^k)$, pk_R^*, ID_R^*;, $IV_1 = 0^r, IV_2 = 0^c$, $IV_3 = IV_2 \oplus 1$. $\mathcal{L}_c = \{IV_2, IV_3\}$

(AFS): $K_h^* \xleftarrow{\$} \{0,1\}^k$; $S^{1*} \xleftarrow{\$} \{0,1\}^\ell$, $S^{2*} \xleftarrow{\$} \{0,1\}^{\ell_{sg}}$

$Y_1^* = \mathsf{Enc}(S^{1*})$,$\sigma^* = \mathsf{Sign}_{\mathsf{sk}_S^*}(S^{2*})$, $T_k^* \xleftarrow{\$} \{0,1\}^k$

$y_1||\ldots||y_e||y_{e+1}||\ldots||y_j = S^{1*}||Y_1^*$;

$K^*||* = \pi_{enc}(\ldots\pi_{enc}(\pi_{enc}(y_1 \oplus IV_1||IV_3) \oplus y_2||0^c)\ldots \oplus y_j||0^c) \oplus K_h^*||0^{b-k}$

On Encryption-Query(M_d)

1. Return:
   $(K_h^*, Y_1^*, Y_2^* = (S^{2*}, \sigma^*), T_k^*)$

On $\pi_{\mathcal{A}}$**-Query** m

1. If $(m = ID_S^*||ID_R^*||K^*||IV_2)$
   then $Bad_K \leftarrow true$
2. $v = \pi(m)$
3. $I_\pi^{\mathcal{A}} = I_\pi^{\mathcal{A}} \bigcup \{(m, v)\}$
4. return v;

On $\pi_{\mathcal{A}}^{-1}$**-Query** v

1. $m = \pi^{-1}(v)$
2. If $(m = ID_S^*||ID_R^*||K^*||IV_2)$
   then $Bad_K \leftarrow true$
3. $I_\pi^{\mathcal{A}} = I_\pi^{\mathcal{A}} \bigcup \{(m, v)\}$
4. return v;

Rest of Oracles same as G10

Figure 7.18: Game G11: All values of encryption oracle replaced by random

Game G9 and Game G10: In G9, K^* is generated randomly. In G10, K^* is computed using the value of randomly generated S^{1*}), K_h^* and Y_1^* . The value of K^* is calculated via $H^{\pi_{Enc}}(IV_1||IV_2, y_1^*, y_2^*, \ldots, y_j^*) \oplus K_h^*$, where $y_1^*, y_2^*, \ldots, y_j^* = S^{1*}||Y_1^*$. Here, $H^{\pi_{enc}}(*)$ represent Sponge function with $IV = IV_1||IV_2$ using permutation π_{enc}. Since π is an ideal permutation and K_h^* is a random value, K^* will also be random. Therefore, G9 and G10 are same.

$$|Pr[EXP_{G10,\mathcal{A}}] = Pr[EXP_{G9,\mathcal{A}}]|.$$

.

Game 11: In Game 10, during signcryption $(K_h^*, S^{1*}, S^{2*}, T^*)$ was calculated using K^* and r-bit random output of π. In Game 11, we directly allocate random $K_h^*, S^{1*}, S^{2*}, T_k^*$ values to signcryption oracle. Earlier in Game 10, during signcryption $(K_h^*, S^{1*}, S^{2*}, T_k^*)$ has a relation with K^*, whereas in G11 there is no relation between $(K_h^*, S^{1*}, S^{2*}, T_k^*)$ and K^*. This gap can be exploited only if K^* is known to adversary $\mathcal{A}$ and queried $ID_S^*||ID_R^*||K^*||IV_2$ to π. We mark this query by $\mathcal{A}$ to π as $Bad_K \leftarrow true$.

Therefore, $|\Pr[EXP_{G11,\mathcal{A}}] - \Pr[EXP_{G10,\mathcal{A}}]| \leq \Pr[Bad_K \leftarrow true]$. If this Bad_K event does not happen then essentially $K_h^*, S^{1*}, S^{2*}, T_k^*$ will be random and also independent from M_d.

Game G12 is the final game of adversary $\mathcal{A}$. It is same as G11, if Bad_K does not happen then essentially S^{1*} remains unknown to adversary along with K^*. Bad_K event in G11 is same as $Bad1_K$ in G12. Because sign-ciphertext is random and independent of M_d, therefore

$|\Pr[EXP_{G12,\mathcal{A}}] = \Pr[EXP_{G11,\mathcal{A}}] = \frac{1}{2}|$

G12: Initialize $I_{enc} = I_\pi = I_\pi^{\mathcal{A}} = \emptyset$, $\mathsf{GenEnc}(1^k)$, pk_R^*, ID_R^*;, $IV_1 = 0^r, IV_2 = 0^c$, $IV_3 = IV_2 \oplus 1$. $\mathcal{L}_c = \{IV_2, IV_3\}$
(AFS): $K_h^* \xleftarrow{\$} \{0,1\}^k$; $S^{1*} \xleftarrow{\$} \{0,1\}^\ell$, $S^{2*} \xleftarrow{\$} \{0,1\}^{\ell_{sg}}$
$Y_1^* = \mathsf{Enc}_{pk}(S^{1*})$,$\sigma^* = \mathsf{Sign}_{\mathsf{sk}_S^*}(S^{2*})$, $T_k^* \xleftarrow{\$} \{0,1\}^k$
$y_1 || \dots || y_e || y_{e+1} || \dots || y_j = S^{1*} || Y_1^*$;
$K^* || * = \pi_{enc}(\dots \pi_{enc}(\pi_{enc}(y_1 \oplus IV_1 || IV_3) \oplus y_2 || 0^c) \dots \oplus y_j || 0^c) \oplus K_h^* || 0^{b-k}$

On Encryption-Query(M_d)

1. Return: $(K_h^*, Y_1^*, Y_2^* = (S^{2*}, \sigma^*), T_k^*)$

On $\pi_{\mathcal{A}}$-Query m

1. If $(m = ID_S^* || ID_R^* || K^* || IV_2)$ then $Bad_K \leftarrow true$
2. $v = \pi(m)$
3. $I_\pi^{\mathcal{A}} = I_\pi^{\mathcal{A}} \bigcup \{(m, v)\}$
4. return v;

On $\pi_{\mathcal{A}}^{-1}$-Query v

1. $m = \pi^{-1}(v)$
2. If $(m = ID_S^* || ID_R^* || K^* || IV_2)$ then $Bad_K \leftarrow true$
3. $I_\pi^{\mathcal{A}} = I_\pi^{\mathcal{A}} \bigcup \{(m, v)\}$
4. return v;

Rest of Oracles same as G11
Red color line shows lines which are not detectable by Adversary.

Adversary $\mathcal{C}$: Given random $Y_1 \xleftarrow{\$} \{0,1\}^{\ell + co_{pe}}$, find S^1 such that $\mathcal{O}^{PC}(S^1, Y_1) = 1$

Game G12 as Adversary A: Initialize $I_{enc} = I_\pi = I_\pi^{\mathcal{A}} = \emptyset$, $\mathsf{GenEnc}(1^k)$, pk_R^*, ID_R^*;, $IV_1 = 0^r, IV_2 = 0^c$, $IV_3 = IV_2 \oplus 1$. $\mathcal{L}_c = \{IV_2, IV_3\}$
(AFS): $K_h^* \xleftarrow{\$} \{0,1\}^k$; $\sigma^* = \mathsf{Sign}_{\mathsf{sk}_S^*}(S^{2*})$, $T_k^* \xleftarrow{\$} \{0,1\}^k$, $Y_1^* = Y_1$

Rest of Oracles same as G12

Finalization: if $\{(u_{1_i} || u_{2_i}), (z_{1_i+1} || z_{2_i+1})\} \in I_\pi^{\mathcal{A}}$ for $i : 1 \to e \to j$ such that $a_i = u_{1_i} \oplus z_{1_i}$, $u_{2_i} = z_{2_i}$ and $\mathcal{O}^{PC}(S^1, Y_1) = 1$, where $S^1 = a_1 || \dots || a_e$, $Y_1 = a_{e+1} || \dots || a_j$, $u_{2_1} = IV_3$ and $z_{1_1} = IV_1$.
then return S^1;

Figure 7.19: Game G12 as final game, and Adversary $\mathcal{C}$ using G12 as Adversary $\mathcal{A}$

The probability of $Bad1_K$ is as follows.

$$\begin{aligned}\Pr[Bad1_K] &= Pr[ID_S^*||ID_R^*||K^*||IV_2 \textit{ is queried to } (\pi_{\mathcal{A}} \textit{ or } \pi_{\mathcal{A}}^{-1})] \\ &= \Pr[(ID_S^*||ID_R^*||K^*||IV_2 \textit{ is queried to } (\pi_{\mathcal{A}} \textit{ or } \pi_{\mathcal{A}}^{-1})) \wedge (I_{enc} \subset I_\pi^{\mathcal{A}})] \\ &\quad + \Pr[(ID_S^*||ID_R^*||K^*||IV_2 \textit{ is queried to } (\pi_{\mathcal{A}} \textit{ or } \pi_{\mathcal{A}}^{-1})) \wedge (I_{enc} \not\subset I_\pi^{\mathcal{A}})].\end{aligned}$$

$(I_{Enc} \subset I_\pi^{\mathcal{A}})$ implies that all the input-output relations of π_{Enc} are also known to the adversary $\mathcal{A}$ via set $I_\pi^{\mathcal{A}}$. Therefore $\mathcal{A}$ knows all y_i^* for $1 \le i \le e$ and h^*. Moreover, the adversary $\mathcal{A}$ learns K^* from K_h^* of challenge ciphertext.

Given $(K_h^*, Y_1^*, Y_2^* = (S^{2*}, \sigma^*), T^*)$, if $ID_S^*||ID_R^*||K^*||IV_2$ is queried to π, then it reveals S^{1*} completely. Therefore,

$$\begin{aligned}\Pr[Bad2] \le Adv_{\text{Encrypt}}^{OW-PCA}(\mathcal{B}_{\mathcal{A}}) + \Pr[(K^*||IV_2 \text{ is queried to } (\pi_{\mathcal{A}} \\ \text{or } \pi_{\mathcal{A}}^{-1})) \wedge (I_{Enc} \not\subset I_\pi^{\mathcal{A}})].\end{aligned}$$

$I_{Enc} \not\subset I_\pi^{\mathcal{A}}$ implies that one of the inputs to $H^{\pi_{Enc}}()$ is unknown to the adversary $\mathcal{A}$. It results in unknown output value from $H^{\pi_{Enc}}()$. Since K_h^* is already random therefore K^* remains unknown and random to $\mathcal{A}$. ID^* and IV_2 are public, therefore, $ID_S^*||ID_R^*||K^*||IV_2$ query to $\pi_{\mathcal{A}}$ is equivalent to random guessing of K^*.

$$\Pr[\text{Bad2}] \le Adv_{\text{Encrypt}}^{OW-PCA}(\mathcal{B}_{\mathcal{A}}) + \frac{(q_{\pi_{\mathcal{A}}} + q_{\pi^{-1}})}{min(2^k, 2^c)}.$$

The last game G12 can be used to simulate adversary $\mathcal{B}$ for simulating adversary $\mathcal{A}$'s queries. Here adversary tries to recover first k-bits from input to Enc on given random y and other public information.

□

Following proof of Lemma 2, we can have following corollaries

Corollary 4. *If the encryption scheme is* OW-PCA*, and the signature scheme is deterministic, then the parallel signcryption scheme is* IND-CCA*.*

This corollary 4 follows directly from lemma 2.

Corollary 5. *If the encryption scheme is deterministic* OW-CPA *and the signature scheme is deterministic, then the parallel signcryption scheme is* IND-CCA*.*

Sign(↓) \ Encrypt(→)		OW-PCA
Deterministic	uUF-RMA	IND-CCA
	suUF-RMA	IND-CCA
Probabilistic	uUF-RMA	✗
	suUF-RMA	IND-CCA

Table 7.3: Privacy of SIGNCRYPT under different combination of Sign and Encrypt

This corollary 5 follows a subclass result of corollary 4, where deterministic OW-CPA secure encryption scheme also follows OW-PCA.

Next, corollary 6 is an another representation of corollaries 4 and 5, where we say only suUF-RMA signature scheme are valid for security. Because deterministic uUF-RMA secure scheme also follows suUF-RMA.

Corollary 6. *If the encryption scheme is deterministic* OW-CPA *and the signature scheme* suUF-RMA*, then the parallel signcryption scheme is* IND-CCA*.*

Corollary 2 and 4 together gives Theorem 8. Corollary 3 and 6 together gives following Theorem 9.

A summary of corollaries related to privacy proof of SIGNCRYPT is summarized in table 7.3.

A gap in results, where probabilistic Sign following uUF-RMA does not provide security to SIGNCRYPT will be addressed in next section.

Theorem 9. *If the encryption scheme is deterministic* OW-CPA*, and the signature scheme is* suUF-RMA*, then the parallel signcryption scheme is secure(*IND/sUF-AdA*).*

Proof of this theorem follows exactly the proof of theorem 8, except that we now assume that Sign is suUF-RMA secure and Encrypt is also deterministic OW-CPA.

7.4.3 Properties

From efficiency point of view, this scheme is significantly optimal since only one SpWrap function call is required before parallel encryption and signature processes. Only one call to *Sponge* function is required after encryption and signature for small amount of data. The reverse process achieves same kind of optimality. Security requirements of this basic scheme, the encryption Encrypt and the signature scheme SIGN are weak which also make this proposal superior to other available schemes.

7.5 Extension of Parallel Signcryption

In previous Section 7.4, we see two limitation of SIGNCRYPT. First, not supporting probabilistic SIGN where same input can give two or more different signatures, for IND-CCA security. Second, there is a restriction on the maximum message length. In this section, we discuss how to extend usage of Parallel Signcryption SIGNCRYPT in case of probabilistic SIGN and in case for arbitrary long messages.

7.5.1 Using Probabilistic Sign

Probabilistic SIGN: This case is not supported in proposed scheme, because we assumed SIGN is deterministic and for same input two different signatures are not considered. In cases, where a probabilistic SIGN scheme needs to be used then IND-CCA security of SIGNCRYPT will no longer valid under proposed scenario. Because now insider adversary can simply produce another signature σ on S^{2*}, of challenge signed-ciphertext, and submit $K_h^*, Y_1^*, Y_2 = (S^{2*}, \sigma), T^*$ to VerDec. This will leads to knowing d bit of M_d with probability 1 without violating the IND-CCA experiment. This case can be handled easily in two ways.

Solution-1 Changing IND-CCAexperiment to IND-gCCA [3]: Consider challenged signed-ciphertext $K_h^*, Y_1^*, Y_2 = (S^{2*}, \sigma^*), T^*$ as two parts. First as ciphertext $K_h^*, Y_1^*, S^{2*}, T^*$ and second as signature σ^*. Imposing a restriction on adversary, attacking IND-CCAsecurity, that not only challenged signed-ciphertext can not be queried to decryption oracle but also those queries are prohibited which result in same as challenged ciphertext $K_h^*, Y_1^*, S^{2*}, T^*$. A query to VerDec having challenge ciphertext $K_h^*, Y_1^*, S^{2*}, T^*$ could be determined easily by using public key of sender as verification key.

This change in IND-CCA experiment is similar to IND-gCCA proposed in [3]. An et al. [3] proposed this IND-gCCAnotion specifically for signcryption in more formal way to avoid trivial attack discussed above. By following the IND-gCCA security experiment we can propose another corollary from Lemma 2 as follows.

Corollary 7. *If the encryption scheme is* OW-PCA *and the signature scheme is unforgeable, then the parallel signcryption scheme is* IND-gCCA*.*

This corollary can be combined with other proposed corollaries from Lemma 1 and different new results can be claimed.

Solution-2 Include σ also as part of input in *Sponge*: This solution follows similar concept as we have followed in case of proposing SpPad–Pe over $\mathcal{F}-SpAEP$. This inclusion of σ in *Sponge* will bind the σ with a particular K, S^2, like in case of Y_1. Now, above discussed attack will not work, because different σ will lead to different K. This change is more simple compared to IND-gCCA security notion requirement. This change is not included initially in proposed scheme with the intension to keep proof simple and straight. Inclusion and reason of this proposed change helps in understanding about IND-gCCAand Y_1, σ as input to *Sponge*.

7.5.2 Arbitrary long messages

Arbitrary long message can be supported in SIGNCRYPT without any major structure modification. Earlier $S^1||S^2||T = C||T$ when $|C||T| = \ell + \ell_{sg} + k$. If $|C||T| > \ell + \ell_{sg} + k$, then $S^1||C^e||S^2||T = C||T$, where $|S^1| = \ell$, $|S^2| = \ell_{sg}$ and final output of SIGNCRYPT is $(K_h, Y_1, C^e, Y_2 = (S^2, \sigma), T_k)$.

Caution: This is essential that if $C||T > \ell + \ell_{sg} + k$ then $S^1||C^e||S^2||T = C||T$, "*not*" as $S^1||S^2||C^e||T = C||T$, where S^1 is input of Enc and S^2 is input of SIGN. This requirement of perform signing on last part of data arises in signcryption to prevent trivial forgery attack by insider adversary. In cases where Sign is performed on data subsequent to Enc data, like $S^1||S^2||C^e||T = C||T$, then adversary can replace C^e and accordingly T using $\pi^{\mathcal{A}}$, sk and pk of Enc. This modification will lead to a trivial forgery.

From the proof of SpPad–Pe we already knows that C^e as part of output will not affect the IND-CCA security of the scheme. In regards to unforgeability, with above mentioned caution, scheme can safely use the Sign. Unforgeability proof of the scheme in case of long messages will follow exactly like of Lemma 1.

With this proposed change from *Solution 2* and support of long message, we call SIGNCRYPT^G as generic version of SIGNCRYPT. Graphical representation of Generic Signcryption is shown in Fig 7.20.

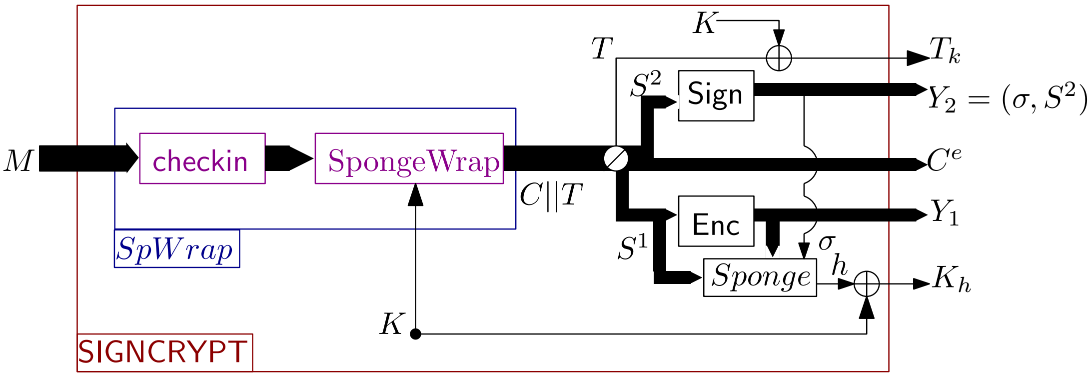


Figure 7.20: Signcryption scheme $\mathsf{SIGNCRYPT}^G$: Input message is passed to *SpWrap*, which uses `checkin` and SpongeWrap function, along with random K. *SpWrap* outputs $C||T$. $C||T$ further split into $S^1||C^e||S^2||T$. Asymmetric encryption scheme Enc take S^1 as input and outputs Y_1. Signature scheme Sign take S^2 as input and output Y_2 which consist of (S^2, σ), where σ is signature. *Sponge* function take Y_1, S^1 and σ as input and outputs h, which further gets xored with K to produce K_h. Final output will be K_h, Y_1, C^e, Y_2, T_k, where $T_k = T \oplus K$.

Theorem 8 can be modified for $\mathsf{SIGNCRYPT}^G$ as follows:

Theorem 10. *If the encryption scheme is OW-PCA, and the signature scheme is (uUF-RMA,suUF-RMA), then the parallel signcryption $\mathsf{SIGNCRYPT}^G$ scheme is secure (IND-CCA/(UF,sUF)-AdA).*

Proof Sketch: If we follow the same proof of Lemma 1, we can clearly see after game G5, output of π is random. Following random π, output h of *Sponge* is also random. Even if adversary tries to use another σ for same S^2, it will result in change of h that leads to random K and T_k, and adversary need to produce target collision over that T or K. This case already included in proof when $I_{vd} \not\subset I_{sc}$ & $I_{vd} \not\subset I_\pi^{\mathcal{A}}$.

For IND-CCA security of $\mathsf{SIGNCRYPT}^G$, we can follow the same proof of Lemma 2 including extra cases when Encrypt and Sign is probabilistic. In order to get information about M_d, now adversary tries to produce different Y_1' for same S^{1*} or different σ for same S^{2*}. Either of these cases will change the value of K^*, which reduces the problem again to having collision on T or having knowledge of S^1. This results in same bound on IND-CCA2 as for SIGNCRYPT.

Therefore, regarding IND-CCA of $\mathsf{SIGNCRYPT}^G$, addition of *Sponge* function is a dummy operation compare to SIGNCRYPT for outputting T, but its usage protect σ of Sign and outputs Y_1 of Encrypt by making them dependent on a particular K. This dependency provides IND-CCA security for $\mathsf{SIGNCRYPT}^G$ in similar way of SIGNCRYPT.

7.6 Conclusion

Combination of encryption and signature scheme yields a signcryption scheme. Extra burden of satisfying both privacy and unforgeability against insider attackers increases the complexity of proving the system secure and efficient. This complexity brings limitation of signcryption scheme in terms of needed security assumption, security achievement and efficiency to balance each other. Message pre-processing is found to be an attractive way to build a secure and efficient signcryption scheme. Already existing message pre-processing techniques are found to be inflexible which disallow their improvement in different case scenarios such as long message length, different type of underlying encryption and signature schemes, insider security, efficient computation in parallel, etc.

Versatile nature of Sponge structure enable us to modify message pre-processing efficiently. Proposed Sponge based message pre-processing helps us to build a secure signcryption scheme achieving higher security level using weak secure encryption and signature scheme. We also found probabilistic or deterministic nature of signature scheme plays an important role in privacy of signcryption scheme but same is not true in case of unforgeability of scheme with respect to underlying encryption scheme. Finally, we are able to find a signcryption scheme that can perform efficiently without compromising its security. Proposed scheme is highly customizable as it allows to use weakly secure and different type of underlying encryption and signature schemes.

Chapter 8

Conclusions

Contents

We now summarize the thesis and discuss some possible future research directions.

8.1 Summary

To convert any weakly secure asymmetric one-way (OW) cryptosystem (Pe) into highly secure PKE which provides indistinguishability against chosen ciphertext attacks (IND-CCA), a basic functionality we require is to randomize the input of Pe and also provide some extra output to recover the used randomness. A message pre-processing serves the purpose of randomizing the input to Pe. This message pre-processing is also known as message padding for asymmetric encryption. First such example was "optimal asymmetric encryption padding" (OAEP). OAEP was found to be useful with RSA, where RSA is deterministic one-way asymmetric primitive (trapdoor one-way permutations). RSA-OAEP has been used in PKCS #1 2.0 standard for a long time. OAEP was found to be quite useful in case of hybrid encryption, signcryption, hybrid signcryption and also as randomness recovery scheme. With time, several schemes were proposed which modified this OAEP. These proposals give different OAEP versions which differ in efficiency, provable security, compatibility with a type of asymmetric one-way cryptosystem (permutation or functions), extending the use of OAEP in other applications, etc. A typical OAEP structure uses some hash functions, working on different input-output setting, in a multi-round Feistel type structure. For having long message support in the asymmetric scheme, a symmetric encryption scheme is combined with OAEP type padding.

In this work, we show that instead of having many different functions to build such an OAEP type scheme, we just require any ideal primitive using which we could build a one-time secure encryption scheme to randomize the input of Pe and a hash function to bind the used randomness. We consider this framework as a generic framework for OAEP-type schemes.

We used Sponge permutation to instantiate the generic framework. Using Sponge permutation, we built a one-time secure encryption scheme and a hash function as a part of padding scheme. We called this padding scheme "Sponge based asymmetric encryption padding" (SpAEP). The versatile and modular nature of Sponge structure allowed us to achieve properties like low ciphertext overhead and support for arbitrary long messages without any additional effort. We are also able to propose a key encapsulation mechanism for hybrid encryption using SpAEP with any trapdoor one-way permutation. SpAEP utilizes the permutation model efficiently in the setting of public key encryption in a novel manner.

Probabilistic nature of the asymmetric one-way primitive (e.g., ElGamal) was found to be incompatible with the OAEP-type schemes, and same happens with SpAEP. Modularity of Sponge structure allowed us to modify the SpAEP into new modified Sponge based padding as SpPad–Pe where SpPad–Pe stands for Sponge based Padding (SpPad) with asymmetric one-way cryptosystem (Pe). SpPad found to be compatible with both deterministic (e.g., RSA) and probabilistic (e.g., ElGamal) functions along with further efficiency improvement compare to SpAEP.

We found the generic structure of OAEP-type scheme, consist of a one-time authentication encryption scheme and a hash function, results in generic strongly secure asymmetric encryption schemes using weakly secure asymmetric one-way cryptosystem. Instead of using specific Sponge based construction we successfully introduced a more generic framework to build a CCA-secure PKE, called REAL. REAL stands for Real time CCA-secure Encryption for Arbitrary Long Messages.

An asymmetric one-way primitive, a one-time secure symmetric encryption scheme and two hash functions are sufficient for this design. Proposed design provides the streaming option without compromising other valuable features, compared to previous works.

We exploit versatile nature of Sponge construction into another area of cryptography known as signcryption, where "Commit-then-Sign&Encrypt" (CtS&E) composition method allows to perform encryption and signing in parallel. We put forward the application of Sponge structure based message padding as an alternative commitment scheme in constructing signcryption schemes. Versatile nature of Sponge structure enables us to modify message pre-processing efficiently. This efficient message padding helps us to achieve a secure signcryption scheme having higher security level using weak secure encryption and signature scheme. We also found nature of signature scheme as probabilistic or deterministic plays an important role in the privacy of signcryption scheme, same is also true in the case of unforgeability when considering nature of encryption scheme. In the end, we were able to find a highly customizable signcryption scheme that can perform efficiently without compromising its security under different nature of encryption and signature schemes.

8.2 Future Directions

As a future work it is worthwhile to investigate following directions:

1. *Ideal model to standard model*: In this thesis, we consider an ideal permutation model for security proof purpose. A practical gap of theory and practice arise when these ideal objects (here ideal permutation) needs to be instantiated. A performance and security gap happens because these instantiated versions do not achieve full ideal behavior that was considered during security proof. For more practical scheme, a scheme proved in standard model is preferred, but these scheme have their own complex nature. A recent proposal of the notion of a public-seed pseudorandom permutation (psPRP) for security assumption on permutations in standard model is proposed in [99].

   A formalization of the schemes proposed in this work under psPRP assumption on used permutation could provide more practically oriented results. These results could bring the results of this thesis closer to practical instantiations, whereas current results of thesis provide theoretical ground.

2. IND-CCA secure PKEs can be designed using different approaches. There are three approaches that deserve our attention. The first applies identity-based-encryption (IBE) techniques, which allow to transform a selective-ID CPA-secure IBE into a CCA-secure PKE [30, 32, 34, 37]. The second approach is based on the concept of lossy trapdoor function introduced by Peikert [85] and further extended by Rosen and Segev [96]. The third approach uses verifiable broadcast encryption, which is proposed by Hanaoka and Kurosawa [59].

   Most of the proposed schemes under these approaches use a specific asymmetric one-way trapdoor cryptosystem like those based on discrete log problems. A generic approach and importance of message pre-processing through message padding could also provide a fruitful direction to this research work. Just as the modularity of Sponge structure has allowed us to use a common padding for both encryption and signature scheme in signcryption, similar facts could also be exploited in aforementioned approaches to construct IND-CCA-secure PKEs.

3. Security of PKE schemes under different security scenarios namely "Selective opening attack", "Key dependent message security" and "Leakage resilient public key cryptography" have provided a broad scope of applicability. Instead of designing a scheme for specific traditional security scenarios, such schemes are preferred which can withstand attack models mentioned before. This helps in having a scheme with multi-faced security and applications.

   Behavior of security and performance of the schemes proposed in this thesis can also be studied under these attack models. It would be interesting to see if the proposed schemes can also withstand these attack models.

4. Redundancy free approach: A common practice in constructing IND-CCA secure scheme is to have a redundancy string which helps in rejecting invalid ciphertexts. Presence of redundancy string in form of MAC, some constant string or in any other form provides an easier way to construct the scheme, simpler security proof and more robust system. However, presence of redundancy string introduces ciphertext overhead which is found to be

non-favorable for constrained bandwidth networks. In case of redundancy free approach, every ciphertext is a valid ciphertext which brings difficulty in simulating decryption function correctly in IND-CCA-security proof. A few works [9, 31, 86] have been done where a IND-CCA secure scheme is proposed without using any redundancy string.

It would be interesting to see if the schemes proposed in this thesis could be converted into a redundancy free scheme without losing security and efficiency features.

Bibliography

[1] M. Abe, R. Gennaro, and K. Kurosawa. Tag-KEM/DEM: A New Framework for Hybrid Encryption. *J. of Cryptology*, 21(1):97–130, 2008.

[2] M. Abe, E. Kiltz, and T. Okamoto. Chosen Ciphertext Security with Optimal Ciphertext Overhead. *IEICE Transactions*, 93-A(1):22–33, 2010.

[3] J. H. An, Y. Dodis, and T. Rabin. On the security of joint signature and encryption. In *EUROCRYPT 2002, Amsterdam, The Netherlands*. Springer, 2002.

[4] J.-P. Aumasson. Password Hashing Competition (PHC), 2014. `https://password-hashing.net/index.html`.

[5] P. Baecher, C. Brzuska, and A. Mittelbach. Reset Indifferentiability and Its Consequences. In *ASIACRYPT 2013, Bengaluru, India*. Springer, 2013.

[6] J. Baek, R. Steinfeld, and Y. Zheng. Formal proofs for the security of signcryption. In *PKC 2002, Paris, France*. Springer, 2002.

[7] J. Baek, W. Susilo, J. K. Liu, and J. Zhou. A New Variant of the Cramer-Shoup KEM Secure against Chosen Ciphertext Attack. In *ACNS 2009, Paris-Rocquencourt, France*. Springer, 2009.

[8] T. K. Bansal, D. Chang, and S. K. Sanadhya. Sponge Based CCA2 Secure Asymmetric Encryption for Arbitrary Length Message. In *ACISP 2015, Brisbane, QLD, Australia*. Springer, 2015.

[9] G. Barthe, D. Pointcheval, and S. Z. Béguelin. Verified security of redundancy-free encryption from Rabin and RSA. In *CCS'12, Raleigh, NC, USA*. ACM, 2012.

[10] M. Bellare, A. Boldyreva, and A. Palacio. An uninstantiable random-oracle-model scheme for a hybrid-encryption problem. In *EUROCRYPT 2004, Interlaken,Switzerland*. Springer, 2004.

[11] M. Bellare, A. Desai, D. Pointcheval, and P. Rogaway. Relations Among Notions of Security for Public-Key Encryption Schemes. In *CRYPTO 1998, Santa Barbara, California, USA*. Springer, 1998.

[12] M. Bellare and P. Rogaway. Random Oracles Are Practical: A Paradigm for Designing Efficient Protocols. In *ACM-CCS 1993, Fairfax, Virginia, USA*. ACM, 1993.

[13] M. Bellare and P. Rogaway. Optimal Asymmetric Encryption. In *EUROCRYPT 1994, Perugia, Italy,*. Springer, 1994.

[14] M. Bellare and P. Rogaway. The exact security of digital signatures - how to sign with RSA and rabin. In *EUROCRYPT 1996, Saragossa, Spain*. Springer, 1996.

[15] M. Bellare and P. Rogaway. Code-Based Game-Playing Proofs and the Security of Triple Encryption, 2004. `http://eprint.iacr.org/2004/331`.

[16] M. Bellare and P. Rogaway. The Security of Triple Encryption and a Framework for Code-Based Game-Playing Proofs. In *EUROCRYPT 2006, St. Petersburg, Russia*. Springer, 2006.

[17] R. Bendlin, S. Krehbiel, and C. Peikert. How to Share a Lattice Trapdoor: Threshold Protocols for Signatures and (H)IBE. In *ACNS 2013, Banff, AB, Canada*. Springer, 2013.

[18] D. Bernstein. Competition for Authenticated Encryption: Security, Applicability, and Robustness(CAESAR), 2017. `https://competitions.cr.yp.to/caesar.html`.

[19] G. Bertoni, J. Daemen, M. Peeters, and G. V. Assche. The Sponge Functions Corner. `http://sponge.noekeon.org/`.

[20] G. Bertoni, J. Daemen, M. Peeters, and G. V. Assche. Sponge functions, 2007. ECRYPT Hash Function Workshop.

[21] G. Bertoni, J. Daemen, M. Peeters, and G. V. Assche. Duplexing the Sponge: Single-Pass Authenticated Encryption and Other Applications. In *SAC 2011, Toronto, ON, Canada*. Springer, 2011.

[22] G. Bertoni, J. Daemen, M. Peeters, and G. V. Assche. Permutation-based encryption, authentication and authenticated encryption, 2012. Directions in Authenticated Ciphers.

[23] G. Bertoni, J. Daemen, M. Peeters, and G. V. Assche. Keccak. In *EUROCRYPT 2013, Athens, Greece*. Springer, 2013.

[24] T. E. Bjørstad and A. W. Dent. Building Better Signcryption Schemes with Tag-KEMs. In *PKC 2006, New York, NY, USA*. Springer, 2006.

[25] T. E. Bjørstad, A. W. Dent, and N. P. Smart. Efficient KEMs with Partial Message Recovery. In *Cryptography and Coding 2007, Cirencester, UK*. Springer, 2007.

[26] D. Bleichenbacher. Chosen ciphertext attacks against protocols based on the RSA encryption standard PKCS #1. In *CRYPTO 1998, Santa Barbara, California, USA*. Springer, 1998.

[27] A. Boldyreva. Strengthening security of RSA-OAEP. In *CT-RSA 2009, San Francisco, CA, USA*. Springer, 2009.

[28] A. Boldyreva, H. Imai, and K. Kobara. How to strengthen the security of RSA-OAEP. *IEEE Trans. Information Theory*, 56(11):5876–5886, 2010.

[29] D. Boneh. Simplified OAEP for the RSA and Rabin Functions. In *CRYPTO 2001, Santa Barbara, California, USA*. Springer, 2001.

[30] D. Boneh and J. Katz. Improved efficiency for cca-secure cryptosystems built using identity-based encryption. In *CT-RSA 2005, San Francisco, CA, USA*. Springer, 2005.

[31] X. Boyen. Miniature CCA2 PK encryption: Tight security without redundancy. In *ASIACRYPT 2007, Kuching, Malaysia*. Springer, 2007.

[32] X. Boyen, Q. Mei, and B. Waters. Direct chosen ciphertext security from identity-based techniques. In *ACM-CCS 2005, Alexandria, VA, USA*. ACM, 2005.

[33] R. Canetti, O. Goldreich, and S. Halevi. The random oracle methodology, revisited. *J. ACM*, 2004.

[34] R. Canetti, S. Halevi, and J. Katz. Chosen-ciphertext security from identity-based encryption. In *EUROCRYPT 2004,Switzerland*. Springer, 2004.

[35] R. Canetti, H. Krawczyk, and J. B. Nielsen. Relaxing Chosen-Ciphertext Security. In *CRYPTO 2003, Santa Barbara, California, USA*. Springer, 2003.

[36] D. Chiba, T. Matsuda, J. C. N. Schuldt, and K. Matsuura. Efficient generic constructions of signcryption with insider security in the multi-user setting. In *ACNS 2011, Nerja, Spain*. Springer, 2011.

[37] S. S. M. Chow, J. K. Liu, and J. Zhou. Identity-based online/offline key encapsulation and encryption. In *ASIACCS '11, Hong Kong, China*. ACM, 2011.

[38] J. Coron. On the Exact Security of Full Domain Hash. In *CRYPTO 2000, Santa Barbara, California, USA*. Springer, 2000.

[39] J. Coron, Y. Dodis, C. Malinaud, and P. Puniya. Merkle-damgård revisited: How to construct a hash function. In *CRYPTO 2005, Santa Barbara, California, USA*. Springer, 2005.

[40] J. Coron, H. Handschuh, M. Joye, P. Paillier, D. Pointcheval, and C. Tymen. GEM: A generic chosen-ciphertext secure encryption method. In *CT-RSA 2002,San Jose, CA, USA*. Springer, 2002.

[41] J. Coron, M. Joye, D. Naccache, and P. Paillier. Universal padding schemes for RSA. In *CRYPTO 2002, Santa Barbara, California, USA*. Springer, 2002.

[42] R. Cramer and V. Shoup. A practical public key cryptosystem provably secure against adaptive chosen ciphertext attack. In *CRYPTO 1999, Santa Barbara, California, USA*. Springer, 1999.

[43] R. Cramer and V. Shoup. Design and Analysis of Practical Public-Key Encryption Schemes Secure against Adaptive Chosen Ciphertext Attack. *IACR Cryptology ePrint Archive*, 2001. `http://eprint.iacr.org/2001/108`.

[44] R. Cramer and V. Shoup. Design and analysis of practical public-key encryption schemes secure against adaptive chosen ciphertext attack. *SIAM J. Comput.*, 33(1):167–226, 2003.

[45] Y. Cui, K. Kobara, and H. Imai. A generic conversion with optimal redundancy. In *CT-RSA 2005,San Francisco, CA, USA*. Springer, 2005.

[46] I. Damgård. A design principle for hash functions. In *CRYPTO 1989, Santa Barbara, California, USA*. Springer, 1989.

[47] A. W. Dent. A Designer's Guide to KEMs. In *Cryptography and Coding, Cirencester, UK*. Springer, 2003.

[48] A. W. Dent. Hybrid signcryption schemes with insider security. In *ACISP 2005, Brisbane, Australia*. Springer, 2005.

[49] A. W. Dent. Hybrid signcryption schemes with outsider security. In *ISC 2005, Singapore*. Springer, 2005.

[50] A. W. Dent and Y. Zheng, editors. *Practical Signcryption*. Information Security and Cryptography. Springer, 2010.

[51] W. Diffie and M. E. Hellman. New directions in cryptography. *IEEE Trans. Information Theory*, 1976.

[52] Y. Dodis, M. J. Freedman, S. Jarecki, and S. Walfish. Versatile padding schemes for joint signature and encryption. In *ACM-CCS 2004, Washington, DC, USA*. ACM, 2004.

[53] Y. Dodis, M. J. Freedman, and S. Walfish. Parallel signcryption with oaep, pss-r, and other feistel paddings, 2003. `http://eprint.iacr.org/2003/043`.

[54] E. Fujisaki and T. Okamoto. Secure integration of asymmetric and symmetric encryption schemes. In *CRYPTO 1999, Santa Barbara, California, USA*. Springer, 1999.

[55] E. Fujisaki and T. Okamoto. Secure integration of asymmetric and symmetric encryption schemes. *J. Cryptology*, 26(1):80–101, 2013.

[56] E. Fujisaki, T. Okamoto, D. Pointcheval, and J. Stern. RSA-OAEP is secure under the RSA assumption. In J. Kilian, editor, *CRYPTO 2001, Santa Barbara, California, USA*. Springer, 2001.

[57] T. E. Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Trans. Information Theory*, 31(4):469–472, 1985.

[58] S. Goldwasser and S. Micali. Probabilistic encryption. *J. Comput. Syst. Sci.*, 28(2):270–299, 1984.

[59] G. Hanaoka and K. Kurosawa. Efficient chosen ciphertext secure public key encryption under the computational diffie-hellman assumption. In *ASIACRYPT 2008, Melbourne, Australia*. Springer, 2008.

[60] J. Hoffstein, J. Pipher, and J. H. Silverman. NTRU: A ring-based public key cryptosystem. In *ANTS, 1998, Oregon, USA*. Springer, 1998.

[61] D. Hofheinz and E. Kiltz. Secure Hybrid Encryption from Weakened Key Encapsulation. In *CRYPTO 2007, Santa Barbara, CA, USA*. Springer, 2007.

[62] E. Kiltz. Chosen-Ciphertext Security from Tag-Based Encryption. In *TCC 2006, New York, NY, USA*. Springer, 2006.

[63] K. Kobara and H. Imai. OAEP++ : A very simple way to apply OAEP to deterministic OW-CPA primitives, 2002. `http://eprint.iacr.org/2002/130`.

[64] N. Koblitz. Elliptic curve cryptosystems. *Mathematics of Computation*, 48(177):203–209, 1987.

[65] Y. Komano and K. Ohta. Efficient universal padding techniques for multiplicative trapdoor one-way permutation. In *CRYPTO 2003, Santa Barbara, California, USA*. Springer, 2003.

[66] K. Kurosawa and Y. Desmedt. A New Paradigm of Hybrid Encryption Scheme. In *CRYPTO 2004, Santa Barbara, California, USA*. Springer, 2004.

[67] K. Kurosawa and Y. Desmedt. A new paradigm of hybrid encryption scheme. In *CRYPTO 2004, Santa Barbara, California, USA*. Springer, 2004.

[68] K. Kurosawa and L. T. Phong. Kurosawa-desmedt key encapsulation mechanism, revisited. In *AFRICACRYPT 2014, Marrakesh, Morocco*. Springer, 2014.

[69] R. Laboratories. PKCS #1 v2.1: RSA cryptography standard, 2002.

[70] G. Leurent and P. Q. Nguyen. How Risky Is the Random-Oracle Model? In *CRYPTO 2009, Santa Barbara, CA, USA*. Springer, 2009.

[71] B. Libert and J. Quisquater. Efficient signcryption with key privacy from gap diffie-hellman groups. In *PKC 2004, Singapore*. Springer, 2004.

[72] J. Malone-Lee and W. Mao. Two birds one stone: Signcryption using RSA. In *CT-RSA 2003, San Francisco, CA, USA*. Springer, 2003.

[73] T. Matsuda, K. Matsuura, and J. C. N. Schuldt. Efficient constructions of signcryption schemes and signcryption composability. In *INDOCRYPT 2009, New Delhi, India*. Springer, 2009.

[74] U. M. Maurer, R. Renner, and C. Holenstein. Indifferentiability, Impossibility Results on Reductions, and Applications to the Random Oracle Methodology. In *TCC 2004, Cambridge, MA, USA*. Springer, 2004.

[75] R. J. McEliece. A public-key cryptosystem based on algebraic. *Coding Thv*, 4244:114–116, 1978.

[76] R. C. Merkle. *Secrecy, Authentication, and Public Key Systems.* PhD thesis, 1979.

[77] D. Micciancio and C. Peikert. Trapdoors for Lattices: Simpler, Tighter, Faster, Smaller. In *EUROCRYPT 2012, Cambridge, UK*. Springer, 2012.

[78] A. Mittelbach. Salvaging Indifferentiability in a Multi-stage Setting. In *EUROCRYPT 2014, Copenhagen, Denmark*. Springer, 2014.

[79] M. Naor and M. Yung. Public-key cryptosystems provably secure against chosen ciphertext attacks. In *STOC 1990, Baltimore, Maryland, USA*. ACM, 1990.

[80] NIST. SHA3 Hash function competition, 2007. `http://csrc.nist.gov/groups/ST/hash/sha-3/index.html`, Last Visited 02-Jan-2017.

[81] T. Okamoto. Authenticated Key Exchange and Key Encapsulation in the Standard Model. In *ASIACRYPT 2007, Kuching, Malaysia*. Springer, 2007.

[82] T. Okamoto and D. Pointcheval. The gap-problems: a new class of problems for the security of cryptographic schemes. In *PKC 2001, Cheju Island, Korea*. Springer, 2001.

[83] T. Okamoto and D. Pointcheval. REACT: rapid enhanced-security asymmetric cryptosystem transform. In *CT-RSA 2001, San Francisco, CA, USA*. Springer, 2001.

[84] C. Peikert. Lattice Cryptography for the Internet. In *PQCrypto 2014, Waterloo, ON, Canada*. Springer, 2014.

[85] C. Peikert and B. Waters. Lossy trapdoor functions and their applications. In *ACM Symposium on Theory of Computing, Victoria, British Columbia, Canada*. ACM, 2008.

[86] D. H. Phan and D. Pointcheval. Chosen-Ciphertext Security without Redundancy. In *ASIACRYPT 2003, Taipei, Taiwan*. Springer, 2003.

[87] D. H. Phan and D. Pointcheval. OAEP 3-Round: A Generic and Secure Asymmetric Encryption Padding. In *ASIACRYPT 2004, Jeju Island, Korea*. Springer, 2004.

[88] J. Pieprzyk and D. Pointcheval. Parallel authentication and public-key encryption. In *ACISP 2003, Wollongong, Australia*. Springer, 2003.

[89] J. Pieprzyk and D. Pointcheval. Parallel signcryption. In *Practical Signcryption*, pages 175–192. Springer, 2010.

[90] D. Pointcheval. Chosen-ciphertext security for any one-way cryptosystem. In *PKC 2000, Melbourne, Victoria, Australia*. Springer, 2000.

[91] M. O. Rabin. Digitalized signatures. In *FOCS 1978, New York, USA*. Academic Press, 1978.

[92] C. Rackoff and D. R. Simon. Non-Interactive Zero-Knowledge Proof of Knowledge and Chosen Ciphertext Attack. In *CRYPTO 1991, Santa Barbara, California, USA,*. Springer, 1991.

[93] T. Ristenpart, H. Shacham, and T. Shrimpton. Careful with Composition: Limitations of the Indifferentiability Framework. In *EUROCRYPT 2011, Tallinn, Estonia*. Springer, 2011.

[94] R. L. Rivest, A. Shamir, and L. M. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 1978.

[95] P. Rogaway. Nonce-based symmetric encryption. In *FSE 2004, Delhi, India*. Springer, 2004.

[96] A. Rosen and G. Segev. Chosen-ciphertext security via correlated products. In *TCC 2009, San Francisco, CA, USA*. Springer, 2009.

[97] V. Shoup. A proposal for an ISO standard for public key encryption. *IACR Cryptology ePrint Archive*, 2001.

[98] V. Shoup. OAEP Reconsidered. *J. Cryptology*, 15(4):223–249, 2002.

[99] P. Soni and S. Tessaro. Public-seed pseudorandom permutations. In *EUROCRYPT 2017, Paris, France*. Springer, 2017.

[100] R. Steinfeld and Y. Zheng. A signcryption scheme based on integer factorization. In *ISW 2000, Wollongong, NSW, Australia*. Springer, 2000.

[101] C. H. Tan. Signcryption scheme in multi-user setting without random oracles. In *IWSEC 2008, Kagawa, Japan*. Springer, 2008.

[102] G. Yuval. How to swindle RABIN. *Cryptologia*, 3(3):187–191, 1979.

[103] Y. Zheng. Digital signcryption or how to achieve cost(signature & encryption)<<cost(signature) + cost(encryption). In *CRYPTO 1997, London, UK*. Springer, 1997.