

# On All-or-Nothing Transforms and Password-Authenticated Key Exchange Protocols

by

Victor Boyko

B.A., New York University (1996)

S.M., Massachusetts Institute of Technology (1998)

Submitted to the Department of Electrical Engineering and Computer  
Science

in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

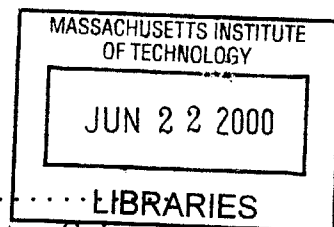
at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2000

© Massachusetts Institute of Technology 2000. All rights reserved.

Author .....  
Department of Electrical Engineering and Computer Science  
May 22, 2000



Certified by .....  
Ronald Rivest  
Professor of Computer Science  
Thesis Supervisor

Accepted by .....  
Arthur C. Smith  
Chairman, Department Committee on Graduate Students



# On All-or-Nothing Transforms and Password-Authenticated Key Exchange Protocols

by

Victor Boyko

Submitted to the Department of Electrical Engineering and Computer Science  
on May 22, 2000, in partial fulfillment of the  
requirements for the degree of  
Doctor of Philosophy

## Abstract

This thesis provides a formal analysis of two kinds of cryptographic objects that used to be treated with much less rigor: All-or-Nothing Transforms (AONTs) and Password-Authenticated Key Exchange protocols. For both, novel formal definitions of security are given, and then practical and efficient constructions are proven secure. The constructions for password-authenticated key exchange are novel, and the AONT construction is an application of an existing scheme to a new area.

AONTs have been proposed by Rivest as a mode of operation for block ciphers. An AONT is an unkeyed, invertible, randomized transformation, with the property that it is hard to invert unless all of the output is known. Applications of AONTs include improving the security and efficiency of encryption. We give several strong formal definitions of security for AONTs. We then prove that Optimal Asymmetric Encryption Padding (OAEP) satisfies these definitions (in the random oracle model). This is the first construction of an AONT that has been proven secure in the strong sense. We also show that no AONT can achieve substantially better security than OAEP.

The second part of this thesis is about password-authenticated key exchange protocols. We present a new protocol called PAK which is the first such Diffie-Hellman-based protocol to provide a formal proof of security (in the random oracle model) against active adversaries. In addition to the PAK protocol that provides mutual *explicit* authentication, we also show a more efficient protocol called PPK that is provably secure in the *implicit*-authentication model. We then extend PAK to a protocol called PAK-X, in which one side (the client) stores a plaintext version of the password, while the other side (the server) only stores a verifier for the password. We formally prove security of PAK-X, even when the server is compromised. Our formal model for password-authenticated key exchange is new, and may be of independent interest.

Thesis Supervisor: Ronald Rivest  
Title: Professor of Computer Science

# Acknowledgments

I would like to start by expressing my deepest gratitude to Ronald Rivest, my thesis advisor, for his constant direction, support, and fruitful discussions. He has been very helpful and supporting throughout my time at MIT. I am also extremely grateful to Shafi Goldwasser and Silvio Micali, members of my thesis committee, for their helpful comments, advice, and discussions.

Research for the first half of this thesis was supported by an NSF Graduate Research Fellowship and DARPA grant DABT63-96-C-0018.

The work for the second half of this thesis was performed in part at Lucent Bell Labs under the direction of and in collaboration with Philip MacKenzie, as well as in collaboration with Sarvar Patel. Another part of the work was supported by an NTT grant. I am also very grateful to Daniel Bleichenbacher for an improvement to our method of generating simulated random oracle responses.

I would like to thank Mihir Bellare, Anand Desai, and Yevgeniy Dodis for notifying me and giving me access to their work on All-or-Nothing Transforms that has appeared after my results. In addition, I would like to thank the following for their help, comments, and advice: Yevgeniy Dodis, Markus Jakobsson, Burt Kaliski, Daniele Micciancio, David Molnar, Kazuo Ohta, Alexander Perlin, Zulfikar Ramzan, Leo Reyzin, Phillip Rogaway, Julien Stern, and Yiannis Tsiounis.

Finally, I would like to thank my parents for making me possible, and for their constant love and encouragement.

# Contents

|          |  |           |
|----------|--|-----------|
| <b>1</b> | <b>General Introduction</b>  | <b>10</b> |
| 1.1      | The Need for Provable Security . . . . .                                 | 10        |
| 1.2      | The Contributions of This Thesis . . . . .                               | 15        |
| <b>2</b> | <b>On the Security Properties of OAEP as an All-or-nothing Transform</b> | <b>16</b> |
| 2.1      | Introduction . . . . .   | 16        |
| 2.1.1    | This Work . . . . .  | 20        |
| 2.1.2    | OAEP . . . . .   | 21        |
| 2.1.3    | Previous Work . . . . .  | 22        |
| 2.1.4    | Subsequent Work . . . . .  | 25        |
| 2.1.5    | Outline . . . . .  | 29        |
| 2.2      | Notation and Model . . . . .   | 29        |
| 2.3      | Definitions . . . . .  | 31        |
| 2.3.1    | Relation of the Definitions to the Applications . . . . .                | 39        |
| 2.4      | Security Results . . . . .   | 41        |
| 2.4.1    | Non-adaptive Indistinguishability: Upper Bound . . . . .                 | 41        |
| 2.4.2    | Non-adaptive Indistinguishability: Lower Bound . . . . .                 | 42        |
| 2.4.3    | Adaptive Indistinguishability . . . . .                                  | 43        |
| 2.4.4    | Non-adaptive Semantic Security . . . . .                                 | 43        |
| 2.4.5    | Adaptive Semantic Security . . . . .                                     | 44        |
| 2.5      | Proofs of Theorems . . . . .   | 44        |
| 2.5.1    | Proof of Theorem 1 . . . . .   | 44        |
| 2.5.2    | Proof Outline for Theorem 2 . . . . .                                    | 55        |

|          |   |           |
|----------|---|-----------|
| 2.5.3    | Proof Outline for Theorem 3 . . . . .   | 56        |
| 2.5.4    | Proof Outline for Theorem 4 . . . . .   | 59        |
| 2.6      | Conclusions and Open Problems . . . . .   | 60        |
| <b>3</b> | <b>Provably Secure Password-Authenticated Key Exchange Using Diffie-Hellman</b> | <b>63</b> |
| 3.1      | Introduction . . . . .  | 63        |
| 3.2      | Background . . . . .  | 65        |
| 3.2.1    | User Authentication . . . . .   | 65        |
| 3.2.2    | Password-Authentication Protocols . . . . .                                     | 67        |
| 3.2.3    | Models for Secure Authentication and Key Exchange . . . . .                     | 68        |
| 3.3      | Model . . . . .   | 69        |
| 3.3.1    | Definition of Security . . . . .  | 69        |
| 3.3.2    | Ideal System . . . . .  | 70        |
| 3.3.3    | Real System with Passwords . . . . .  | 75        |
| 3.4      | Explicit Authentication: The PAK Protocol . . . . .                             | 77        |
| 3.4.1    | Preliminaries . . . . .   | 77        |
| 3.4.2    | The Protocol . . . . .  | 78        |
| 3.5      | Implicit Authentication: The PPK Protocol . . . . .                             | 79        |
| 3.5.1    | Ideal System with Implicit Authentication . . . . .                             | 79        |
| 3.5.2    | PPK Protocol . . . . .  | 80        |
| 3.6      | Resilience to Server Compromise: The PAK-X Protocol . . . . .                   | 82        |
| 3.6.1    | Ideal System with Passwords: Resilience to Server Compromise . . . . .          | 82        |
| 3.6.2    | Real System: Resilience to Server Compromise . . . . .                          | 83        |
| 3.6.3    | PAK-X Protocol . . . . .  | 84        |
| 3.7      | Security of the PAK Protocol . . . . .  | 84        |
| 3.7.1    | The Simulator . . . . .   | 86        |
| 3.7.2    | Proofs of Claims . . . . .  | 97        |
| 3.8      | Security of the PPK Protocol . . . . .  | 103       |
| 3.8.1    | The Simulator . . . . .   | 103       |

|          |  |            |
|----------|--|------------|
| 3.8.2    | Proofs of Claims . . . . .               | 112        |
| 3.9      | Security of the PAK-X Protocol . . . . . | 121        |
| 3.9.1    | The Simulator . . . . .                  | 121        |
| 3.9.2    | Proofs of Claims . . . . .               | 132        |
| 3.10     | Conclusions and Open Problems . . . . .  | 139        |
| <b>4</b> | <b>General Conclusion</b>                | <b>141</b> |

# List of Figures

|     |   |    |
|-----|---|----|
| 2-1 | The use of an AONT against brute force attacks . . . . .            | 17 |
| 2-2 | The use of an AONT for efficient encryption . . . . .               | 18 |
| 2-3 | The use of an AONT for remotely-keyed encryption . . . . .          | 19 |
| 2-4 | A diagram of the OAEP . . . . .                                     | 22 |
| 2-5 | Diagram of the non-adaptive indistinguishability scenario . . . . . | 32 |
| 2-6 | Diagram of the adaptive indistinguishability scenario . . . . .     | 34 |
| 2-7 | Diagram of the non-adaptive semantic security scenario . . . . .    | 37 |
| 2-8 | OAEP and related functions . . . . .                                | 46 |
| 3-1 | The PAK protocol . . . . .  | 78 |
| 3-2 | The PPK protocol . . . . .  | 81 |
| 3-3 | The PAK-X protocol . . . . .  | 85 |

# List of Tables

|     |   |    |
|-----|---|----|
| 3.1 | Valid connection assignments for the <i>start session</i> operation . . . . . | 72 |
| 3.2 | Possible shadowings in the PAK simulator . . . . .                            | 87 |

# Chapter 1

## General Introduction

### 1.1 The Need for Provable Security

*Cryptography*, the art of secret communication, has been in use for thousands of years. From Caesar’s simple substitution system, to more complicated monoalphabetic ciphers, then to polyalphabetics, and on to cipher machines in the first half of the 20th century — the techniques were constantly increasing in complexity. The development of new systems was spurred by the efforts in *cryptanalysis* — the art of breaking the other side’s systems. The first records on the methods of cryptanalysis appear in the 14th and 15th centuries (see Kahn [61] for an extensive history). From that time on, cryptographers and cryptanalysts were engaged in a constant race, with ciphers being designed, broken, redesigned, and so on time and again. This situation continued without much change through World War II, during which cryptanalysis, especially of the German Enigma machine, played a major role.

The state of affairs changed greatly with the widespread use of computers in the 1960s and 1970s. New systems have appeared, such as DES [75], Diffie-Hellman [35], and RSA [89], that seemed to be unbreakable. As Kahn [61] wrote in 1996, “The war of cryptographer against cryptanalyst has been won by the cryptographers” (p. 984). However, even in recent times cryptographic schemes get attacked and broken. Of course, a lot has changed since World War II. The field of cryptography has expanded far beyond mere protection of secrecy of messages, to include such applications as

public key encryption, digital signatures, and key exchange. On the other hand, each new application carries with it its own notion of attack. Here are some examples of cryptanalysis in modern cryptology: In the area of public key encryption, we have the large family of knapsack cryptosystems [71, 28, 27] most of which have been broken [92, 22, 98]. Among signature schemes, let us point out the the Ong-Schnorr-Shamir scheme [80, 81] which was broken [1], fixed and again broken [84], then the cycle repeated itself one more time [39], until finally a new fixed version was made [74]. For key exchange, we have the Needham-Schroeder protocol [77], which has been attacked [33] and fixed [78].

It would seem that cryptographers continued with the old routine: They would design a scheme, and try to break it. If they couldn't break it, they would show the scheme to their friends, and see if they could cryptanalyze it. If that test was passed as well, the scheme would be used (or at least published). The scheme could then be in use for years, until later cryptanalysts would discover new attacks and break it, at which point another scheme would be made, and the cycle would repeat. This arrangement would seem perfect for providing both cryptographers and cryptanalysts with job security, but it is not too satisfactory to the users. The question becomes: What can we do to *today* to protect ourselves against *tomorrow's* attacks?

An approach to this problem was introduced by the notion of *provable security*: we can't predict what an adversary might do, but we can try to *prove* that she won't succeed no matter what strategy she employs. A number of provably secure constructions of cryptographic primitives have been proposed, two of the most prominent ones being the Goldwasser and Micali [46] public-key encryption scheme and the Goldwasser, Micali, and Rivest [48] signature scheme.

When saying that a particular construction is “provably secure,” there are several questions that need to be asked: What exactly is being proven? What, if any, are the assumptions of the proof? To answer the first question, i.e., to precisely formulate the statement that “the adversary won't succeed,” one needs to carefully define a *formal model* of security for the application under consideration. The model needs to precisely specify what exactly the adversary may do (i.e., the attack scenario), as

well as what goal the adversary should not be allowed to achieve. For instance, in the application of encryption, the adversary’s allowed behavior could range from simply observing the ciphertext (a passive attack), to querying the decryption function on the values of the adversary’s choice (adaptive chosen-ciphertext attack). The adversary’s goal could also vary: it could be determining partial information about the message (breaking *semantic security* [46]), distinguishing the encryptions of two messages (breaking *indistinguishability* [46]), or constructing a new plaintext-ciphertext pair with certain properties (breaking *non-malleability* [36]). Some models may be equivalent (the equivalence of indistinguishability and semantic security is shown by Micali et al. [72]), some may be strictly stronger than others (non-malleability with a chosen-plaintext attack (CPA) implies indistinguishability with a CPA, but not vice versa [6]), and some may be uncomparable (non-malleability with a CPA neither implies nor is implied by indistinguishability with a lunchtime chosen-ciphertext attack [6]).

The second major question when talking about “provable security” is the assumptions of the proof. Ideally, we would like to prove security without any special assumptions (this is the so called *information-theoretic* security), but very few results have been obtained in such a model (the one-time pad [99] is a famous example of an information-theoretically secure, yet rather impractical system). Most of modern cryptography is based on assumptions of computational infeasibility of certain tasks. The minimal such assumption seems to be the existence of *one-way functions* (OWFs), which are functions that are easy to compute, yet hard to invert (more precisely, no polynomial-time adversary can invert them with non-negligible probability). OWFs have been shown necessary for the existence of secure private-key encryption [54], pseudorandom generators [64], and digital signatures [91] (as well as other applications). For the three applications just mentioned, provably secure constructions have indeed been proposed based on generic OWFs [45, 53, 91]. However, those constructions are generally too inefficient to use in practice, and appear interesting mainly as existence proofs. There are also limitations on what may be built from OWFs: Impagliazzo and Rudich [55] have provided evidence that certain cryptographic ap-

plications, such as secret key exchange, are unlikely to be achievable using OWFs in a black-box manner.

To construct efficient schemes, as well as to achieve tasks that appear to require more than just generic OWFs, cryptographers have relied on a number of specific computational assumptions, such as the hardness of computing discrete logarithms [69], the Diffie-Hellman problem [35], and the RSA problem [88]. However, the security guarantees of schemes based on these problems have generally been low. For instance, even if we assume that the RSA function  $x \mapsto (x^e \bmod n)$  is hard to invert, that does not guarantee that no partial information about the input will be leaked, when the function is used for encryption in the RSA cryptosystem. In fact, it is known that certain kinds of partial information do get leaked.

More recently, two assumptions have appeared in the literature that lead to the constructions of schemes that are efficient, and yet offer high guarantees of security: the Diffie-Hellman indistinguishability assumption [18] (also known as the DDH assumption), and the random oracle assumption [9]. (We note that both of these assumptions are used in this thesis.) The DDH assumption by itself already implies that the ElGamal public-key cryptosystem [38] is semantically secure (while based on just the regular Diffie-Hellman assumption, we can only say that the scheme is hard to invert). The DDH assumption has also been used to construct an efficient public-key cryptosystem with a very high level of security, namely the Cramer-Shoup system [29], which is provably secure against chosen-ciphertext attacks.

The random oracle model was introduced by Bellare and Rogaway [9]. The idea of the model is that all the participants (including the algorithms and the adversary) have access to the same fixed random function. In a practical implementation, the ideal of a random function would be approximated by a public and deterministic hash function, such as SHA [76]. A number of efficient schemes have been proposed and proven secure in the random oracle model, such as the OAEP method for encryption [10], or the PSS signature scheme [12]. The main disadvantage of the random oracle model is that it is unrealistic, in the sense that no deterministic function could ever have the properties of a random function. In fact, Canetti et al. [24] show that

there are some (admittedly artificial) schemes that are secure in the random oracle model, but not in the standard model, no matter how the random function is instantiated. However, as far as we know, no “reasonable” cryptographic construction has been shown secure in the random oracle model, and yet compromised in the standard model. Clearly, random oracle security is substantially better than just heuristic confidence (as it does rule out “generic” attacks that are independent of the hash function), but relying only on standard complexity assumptions is definitely preferable.

Finally, let us point out some limitations of provable security. First of all, a proof of security is valuable only if the formal model is properly chosen. For instance, the original work on zero-knowledge (ZK) proofs [47] was in the model where there are just two parties. This model is indeed appropriate in some scenarios (such as the host-smartcard scenario). However, it is not suitable for client-server applications, where many clients could be connecting to the server at the same time. In fact, many standard ZK protocols are not provably secure in the concurrent setting. New techniques are required to achieve security in the concurrent model [37, 85]. As another example, let us point out that there are classes of practical attacks that are not addressed by most standard models, such as timing attacks [63] (where the adversary can measure the time taken by a party’s operations), or power analysis attacks [62] (where the adversary can monitor the amount of power consumed by a party’s circuits).

Second, it is important to keep in mind that security based on a certain assumption is only as good as the assumption itself. If the assumption is ever broken (e.g., a sufficiently efficient way is found to solve a problem that was previously believed to be hard), then the scheme might no longer be secure. Also, even if the underlying problem is infeasible asymptotically, it could still be solvable for practical values of parameters. As an example, the Ajtai-Dwork cryptosystem [2] is provably secure based on the difficulty of a certain lattice problem. However, the system was successfully attacked by Nguyen and Stern [79], for practical values of parameters, by applying algorithms that can heuristically solve such lattice problems. We should also

note that even if the underlying problem is hard, the reduction from a scheme’s security to solving the problem may be so inefficient as to make the security guarantees worthless. As a hypothetical example, suppose an attack of time  $T$  on a cryptosystem translated into the ability to find a collision in the SHA function (which has 160 bits of output) in time  $T^5$ . Since a collision in SHA can be found in time  $\sim 2^{80}$  through random sampling (by the birthday paradox), it would follow that the security of the cryptosystem is only  $(2^{80})^{1/5} = 32768$ , which is quite small, even though the security of SHA ( $2^{80}$ ) is substantial.

In summary, although provable security has limitations, it appears to be the only reasonable hope for making schemes that will stay unbreakable for a long period of time. Cryptanalysts will still have what to do, of course, even if they will have to concentrate all their efforts on solving the underlying hard problems. However, barring major progress in those areas, cryptographers will no longer need to go through the old design-break-redesign cycle. We think, then, that the major task in cryptography right now is to construct new schemes that are efficient and provably secure, for more and more applications.

## 1.2 The Contributions of This Thesis

The task of this thesis is to rigorously define, and provide efficient and provably secure constructions, for the following two cryptographic applications: All-or-Nothing Transforms (AONTs) and Password-Authenticated Key Exchange protocols. Both areas are relatively recent, and have not received much previous formal treatment. In fact, this thesis is the first work we are aware of to give strong formal definitions of AONTs, and provide a provably secure construction. In the field of password authentication, we know of only one preceding work based on formal methods [67]. Compared to that work, we provide a novel model of security (which we feel is more elegant), as well as constructions that are more efficient and rely on fewer security assumptions.

## Chapter 2

# On the Security Properties of OAEP as an All-or-nothing Transform

### 2.1 Introduction

The concept of an *All-or-Nothing Transform (AONT)* was introduced by Rivest [86] to increase the cost of brute force attacks on block ciphers without changing the key length. As defined by Rivest [86], an AONT is an efficiently computable transformation  $f$ , mapping sequences of blocks (i.e., fixed length strings) to sequences of blocks, which has the following properties:

- Given all of  $f(x_1, \dots, x_n) = (y_1, \dots, y_{n'})$ , it is easy to compute  $x_1, \dots, x_n$ .
- Given all but one of the blocks of the output (i.e., given  $y_1, \dots, y_{j-1}, y_{j+1}, \dots, y_{n'}$  for any  $1 \leq j \leq n'$ ), it is infeasible to find out any information about any of the original blocks  $x_i$ .

As mentioned by Rivest [86], an AONT should be randomized, so that a known message does not yield a known output.

An AONT itself does not perform any encryption, since there is no secret key information involved. However, if its output is encrypted, block by block, with a block

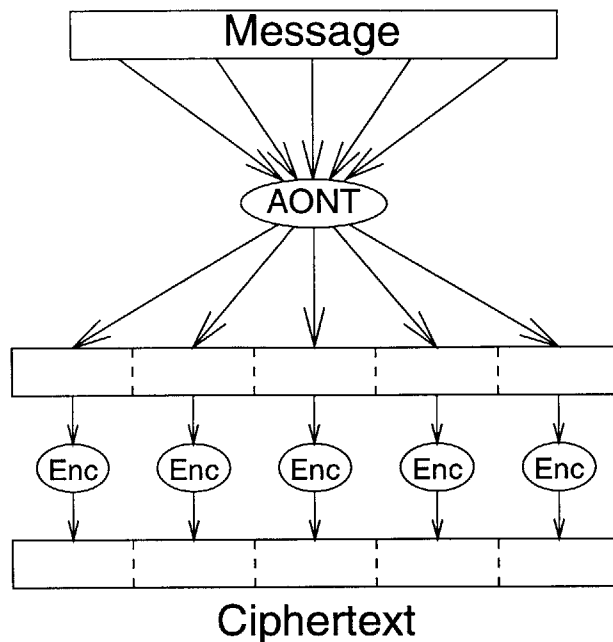


Figure 2-1: The use of an AONT against brute force attacks

cipher (see Figure 2-1), the resulting scheme will have the following interesting property: An adversary cannot find out any information about any block of the message without decrypting all the blocks of the ciphertext. Now, if the adversary attempts to do an exhaustive search for the key, she will need to perform  $n'$  decryptions before determining whether a given key is correct. Thus, the attack will be slowed down by a factor of  $n'$ , without any change in the size of the secret key. This is particularly important in scenarios where the key length is constrained to be insecure or marginally secure (e.g., because of export regulations).

Another very important application of AONTs, as proposed by Johnson et al. [60] for inclusion in the IEEE P1363a standard, is to make fixed-blocksize encryption schemes more efficient. Instead of encrypting the whole message block by block, we can apply AONT to it, and encrypt just some of the blocks of the output (see Figure 2-2). When used with a public key cryptosystem, such as RSA [89], this method can give a substantial reduction in communication overhead, as compared to the traditional approach of generating a random symmetric key, encrypting it

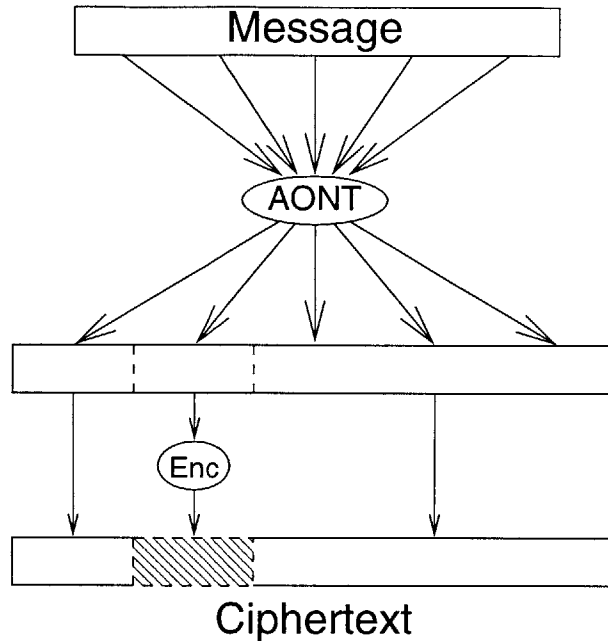


Figure 2-2: The use of an AONT for efficient encryption

with the public key, and then encrypting the message with the symmetric key. For example, the overhead of the traditional method with RSA would be 1024 bits (for the encryption of the random key). On the other hand, the new method, using the AONT construction proposed in this paper, would give an overhead of less than 200 bits. This method is even more useful for elliptic-curve cryptosystems, which typically have a block length that is too small to fit a symmetric key, together with padding and redundancy (see Johnson and Matyas [59]). A similar application of AONTs, as proposed by Rivest [87], would be to reduce communication requirements in a case where the encryption function greatly expands its input.

The use of AONT with encryption can be particularly useful for remotely keyed encryption, i.e., applications where the part of the system that contains the keys is separate, and where bandwidth restrictions prevent us from sending the whole message from the insecure to the secure component (see Blaze [17]). An example of such a scenario would be the case where the keys are stored in a smartcard, and the user wishes to encrypt or decrypt large files. Through the use of AONT, we can

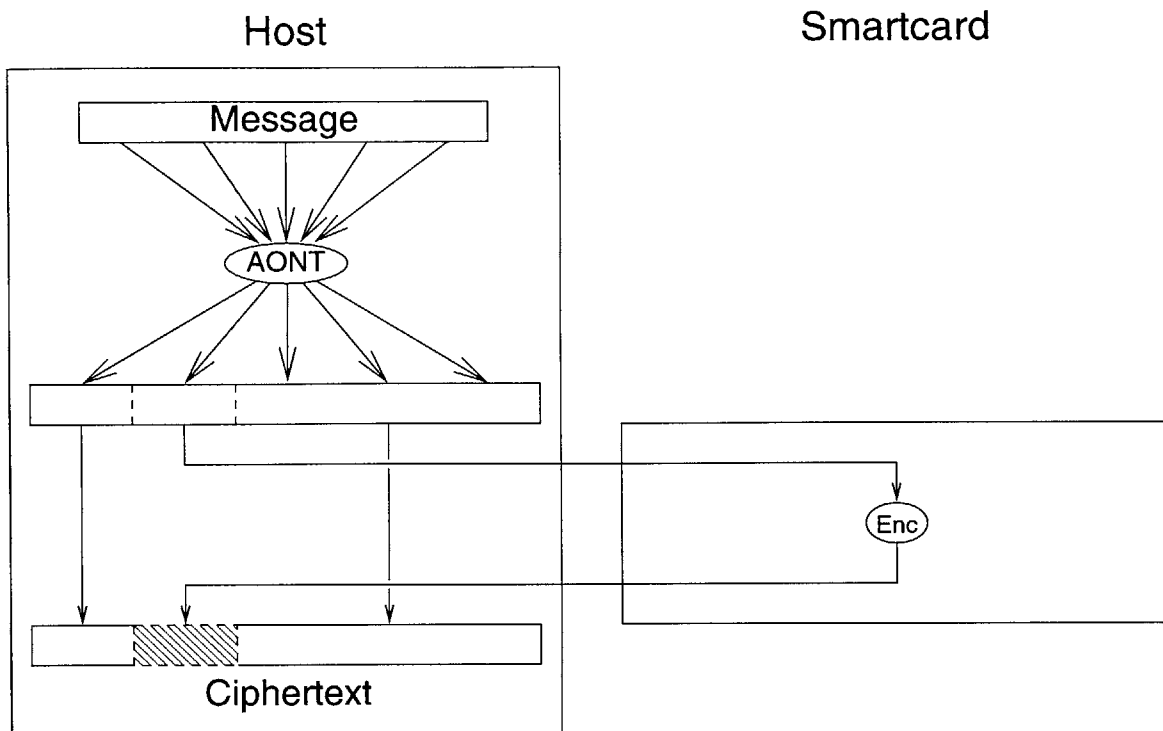


Figure 2-3: The use of an AONT for remotely-keyed encryption

completely eliminate any encryption components from the host system, and restrict such operations to the smart card (this is a generalization of the scheme of Jakobsson et al. [58], substituting general AONTs for the OAEP-like construction used in that paper). The host would transform the message with an AONT, and send one block to the smartcard (see Figure 2-3). The smartcard would encrypt that block, and return it to the host. The encryption of the message will then be the output of the AONT, with one block encrypted. Assuming the block encryption is secure, the whole message will be secure. Note that since the host system does not contain any encryption algorithms, it might not be subject to export regulations.

The major problem with the definition of Rivest [86] is as follows: That definition only speaks about the amount of information that can be learned about a *particular* message block. It does not, however, address the issue of information about the message as a whole (e.g., the XOR of all the blocks). To make the AONT truly useful, we would want it to hide *all* information about the input if any part of the

output is missing (we will refer to this as the *semantic security model*). For instance, if an AONT is used for the purpose of slowing down exhaustive search of the key space, a relation between several blocks of the plaintext may provide enough information to the adversary for the purpose of detecting an invalid key. Similarly, security of encryption should incorporate the adversary’s inability to learn partial information about the plaintext after seeing the ciphertext.

Another disadvantage of the model of Rivest [86] is that it does not consider the relation between the number of bits of AONT output that the adversary has, and the information that is leaked about the input. That model only considers the cases when the adversary has the whole output (in which case she should be able to completely determine the input), and when at least one complete block of the output is missing (in which case it should be infeasible to determine any block of the input). It would be interesting to consider exactly how much information about the input can be determined by looking at all but a certain number  $l$  bits of the AONT output, and how much effort is required to obtain that information.

### 2.1.1 This Work

The goal of this part of the thesis is to provide an AONT construction that is provably secure in the strong sense described above. Our contributions are as follows:

- We give new formal definitions of AONT security in terms of semantic security and indistinguishability. These definitions address the concerns mentioned above and provide the security needed for practical applications. They are parallel to the two notions of security for public-key cryptosystems, defined by Goldwasser and Micali [46]. We consider both the non-adaptive scenario (where the positions of the bits that are removed from AONT output are fixed before the experiment), and the adaptive scenario (where the adversary can choose the positions).
- We prove that OAEP (see Section 2.1.2), a construction originally introduced by Bellare and Rogaway in a different context, satisfies these definitions (in the

random oracle model).

- We give an upper bound on the adversary’s advantage in getting information about OAEP input when given all but  $l$  bits of OAEP output, as opposed to having none of the output. The bound is exact, i.e., does not involve asymptotics. It does not use any computational assumptions and relies only on the properties of random oracles. The bound is directly proportional to the number of adversary’s queries to the random oracle and is inversely exponential in the number of bits of OAEP output that are withheld from the adversary.
- We then show that our upper bound is nearly optimal, in the sense that no adversary can do substantially better against OAEP than by exhaustive search. In addition, it will follow that no AONT can achieve substantially better security (i.e., upper bound on the adversary’s advantage) than OAEP.

### 2.1.2 OAEP

*Optimal Asymmetric Encryption Padding (OAEP)* was originally introduced by Bellare and Rogaway [10] for the purpose of constructing semantically secure and plaintext-aware public-key encryption schemes from arbitrary trapdoor permutations. For parameters  $n$  and  $k_0$ , “generator”  $G : \{0, 1\}^{k_0} \rightarrow \{0, 1\}^n$ , and “hash function”  $H : \{0, 1\}^n \rightarrow \{0, 1\}^{k_0}$ , the transform  $\text{OAEP} : \{0, 1\}^n \times \{0, 1\}^{k_0} \rightarrow \{0, 1\}^{n'}$ , for  $n' = n + k_0$ , is defined as

$$\text{OAEP}^{G,H}(x, r) = x \oplus G(r) \parallel r \oplus H(x \oplus G(r)),$$

where  $\parallel$  denotes concatenation. Here  $x$  is the message and  $r$  is a random string. In applications,  $n$  would be the length of a message, and  $k_0$  would be the security parameter, e.g.,  $k_0 = 128$ . We will often refer to the first half of the OAEP output (i.e.,  $x \oplus G(r)$ ) as  $s$ , and to the second half (i.e.,  $r \oplus H(s)$ ) as  $t$ . Here  $|s| = n$  and  $|t| = k_0$ . We may also write  $\text{OAEP}^{G,H}(x)$ , implying that  $r$  is chosen uniformly at random from  $\{0, 1\}^{k_0}$ .

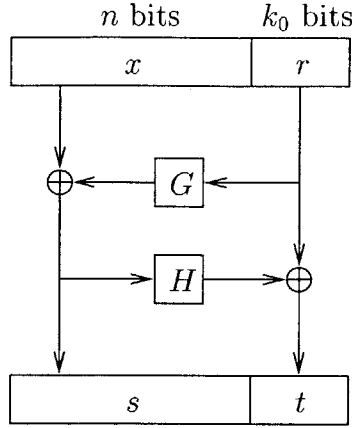


Figure 2-4: A diagram of the OAEP

A diagram of the OAEP appears in Figure 2-4.

Functions  $G$  and  $H$  are “random oracles,” as introduced by Bellare and Rogaway [9]. The same authors show [10] that if  $G$  and  $H$  are “ideal,” i.e., they are random functions, and  $f : \{0, 1\}^{k_0+n} \rightarrow \{0, 1\}^{k_0+n}$  is a trapdoor permutation, then the encryption scheme

$$\mathcal{E}^{G,H}(x) = f(\text{OAEP}^{G,H}(x, r)),$$

with  $r$  chosen at random for each encryption, is semantically secure, in the sense of Goldwasser and Micali [46].

### 2.1.3 Previous Work

Rivest [86] has proposed the following construction (“the package transform”) as a candidate AONT:

- Let  $E$  be a block cipher. Let  $K_0$  be a fixed, publicly known key for  $E$ .
- Let the input message be the sequence of blocks  $m_1, m_2, \dots, m_s$ .
- Choose at random a key  $K'$  for  $E$ .
- Compute the output sequence  $m'_1, m'_2, \dots, m'_{s'}$ , for  $s' = s + 1$ , as follows:

- Let  $m'_i = m_i \oplus E(K', i)$  for  $i = 1, 2, \dots, s$ .
- Let

$$m'_{s'} = K' \oplus h_1 \oplus h_2 \oplus \dots \oplus h_s,$$

where

$$h_i = E(K_0, m'_i \oplus i)$$

for  $i = 1, 2, \dots, s$ .

No formal proof was given that this construction is actually an AONT. The heuristic argument for security is based on the idea that if any block of the output is unknown, then  $K'$  cannot be computed, and so it is infeasible to compute any message block. Rivest [86] mentions that “the package transform” can be viewed as a special case of the OAEP, for  $G(x) = E(x, 1) \| E(x, 2) \| \dots \| E(x, s)$  and  $H(x) = \bigoplus_{i=1}^s E(K_0, x_i \oplus i)$ . However, no claims about OAEP itself are made in that paper.

Johnson et al. [60], in their contribution to the IEEE P1363a standard, give an OAEP-like transform that uses four rounds of hash applications instead of two. A heuristic analysis of the security of that construction is given by Matyas et al. [68]. Using an informal assumption about the hardness of the underlying hash functions, they argue that the number of operations required to determine the secret bits in the input message grows exponentially with the number of unknown bits. However, we are not aware of any formal proof of security of the transform from Johnson et al. [60]. In any case, the analysis of Matyas et al. [68] is not directly applicable if there are fewer than four rounds, so it does not work for OAEP.

Stinson [96] gives a treatment of AONTs from the point of view of unconditional security. Similarly to Rivest [86], Stinson’s definition only considers the amount of information leaked about a particular block of the message, as opposed to the whole message. He uses a straightforward formalization of Rivest’s definition above, suitably modified for information-theoretic security. Stinson then goes on to propose

some constructions for AONTs using linear transforms, which can be proven secure in that model. The basic idea of these constructions is to use the function  $\phi(\mathbf{x}) = \mathbf{x}M^{-1}$ , where  $\mathbf{x}$  is a vector of  $s$  message blocks (considered as elements of  $GF(q)$ , for some prime power  $q$ ), and  $M$  is an invertible  $s$  by  $s$  matrix over  $GF(q)$ , such that no entry of  $M$  is equal to 0. It is easy to see that each component of  $\mathbf{x}$  linearly depends on all the components of  $\mathbf{y} = \phi(\mathbf{x})$  (since  $\mathbf{x} = \mathbf{y}M$ ).

It is conceivable that Rivest’s “package transform” would be secure in the semantic security model (with sufficiently strong assumptions about the block cipher). The construction of Johnson et al. may also be secure, although no formal proof has been given. However, the linear constructions of Stinson would definitely not be secure in that model, since it is easy to come up with linear relations among the elements of  $\mathbf{x}$  by looking at just a few elements of  $\phi(\mathbf{x})$  (in fact, since  $\phi$  is linear and deterministic, *every* output of  $\phi(\mathbf{x})$  gives a linear relation on elements of  $\mathbf{x}$ ). Even if the message is padded with random blocks, it is still possible to extract partial information about the message if the number of known outputs is larger than the number of random blocks.

It is interesting to note that the relationship between the number of missing bits and adversary’s required effort has come up in other contexts. Merkle [70], in one of the first papers on public key cryptography, defines the concept of a “puzzle,” which is a cryptogram that requires  $\Theta(N)$  work to break, where  $N$  is some number depending on the security parameters (the total amount of work put in by the communication parties is going to be  $\Theta(N)$ ). Merkle’s proposed construction of such “puzzles” is to take a block cipher and restrict the size of the key space, by varying only  $\Theta(\log N)$  bits of the key and fixing the rest. It is assumed that breaking a cryptogram of the underlying cipher, when all but  $\Theta(\log N)$  bits of the key are known, requires  $\Theta(N)$  work.

Even et al. [42] assume the existence of a “uniformly secure” block cipher for their construction of a contract signing protocol. They consider a block cipher “uniformly secure” if it is infeasible to find a key for a given plaintext-ciphertext pair when no information about the key is known; but if the first  $i$  bits of the key are known, then

there is an algorithm for breaking the cryptogram in time  $t(k - i)$ , for some function  $t(\cdot)$ , and no algorithm can do it faster than in time  $\frac{1}{2}t(k - i)$ . Here  $k$  is the key length.

Both Merkle and Even et al. conjecture that standard block ciphers, such as Lucifer [43] or DES [75], satisfy their assumptions. However, uniform security is probably not a common consideration in block cipher design, as almost all applications of these primitives assume the whole key to be secret. Thus, it may be unsafe to make such an assumption about standard block ciphers. In fact, this is, in effect, one of the criticisms given by Ben-Or et al. [15] of the work of Even et al. It seems to us, however, that the methods of this thesis can be used to give a simple construction that will turn any block cipher that is secure in the regular sense into one which is uniformly secure. See Section 2.6 for more details.

#### 2.1.4 Subsequent Work

Since the publication of the conference version of these results [20], several further results have appeared that cite this work and consider various variations and applications of AONTs. Canetti et al. [23] generalize our definitions and give some provably secure constructions of AONTs (in their new model) without random oracles. In addition, they prove some general results (such as that the existence of computational AONTs implies the existence of one-way functions). Desai [34] analyzes the security of AONTs for protection against exhaustive key search. Bellare and Boldyreva [4] analyze the security of AONTs (and in particular, OAEP) in the application of chaffing-and-winnowing, with some of their results applicable to general application of encryption.

##### The Work of Canetti et al.

The paper of Canetti et al. [23] can be viewed as the most direct continuation of this work, as it provides a very important next step: constructions and analysis of AONTs without random oracles. We will only speak here about their model and construction (see Section 2.3.1 for the description of some new applications for AONTs proposed

in that paper). They formulate a security definition, which is very similar to the non-adaptive indistinguishability model described in Section 2.3. The only important differences are that no random oracles are used, and the security is described in terms of a threshold: As long as more than a certain number  $\ell$  bits of the output are unavailable to the adversary, the adversary should be unable to find out any information about the input. The adversary's inability to find out information is formulated in terms of either computational, statistical, or perfect indistinguishability of certain distributions. An AONT with threshold  $\ell$  is called an  $\ell$ -AONT. The authors then proceed to give an extension of the model, by allowing an AONT to have a *public* and a *secret* output. The definition of security would then presume that the adversary is always given the public output, and bits can be only be removed from the secret part. As pointed out by Canetti et al., an  $\ell$ -AONT with public and secret outputs of length  $p$  and  $s$ , respectively, also gives a secret-only (i.e., traditional)  $\ell'$ -AONT with output size  $N = s + p$  and  $\ell' = \ell + p$  (since if the adversary misses  $\ell + p$  bits of the output, that means it must miss at least  $\ell$  bits of the secret output).

The authors then present a construction based on a new primitive, called an *Exposure Resilient Function* (ERF): A function is an  $\ell$ -ERF if its outputs are (perfectly, statistically, or computationally) random even if all but  $\ell$  bits of the input are known. Given an  $\ell$ -ERF  $f : \{0, 1\}^n \rightarrow \{0, 1\}^k$ , an  $\ell$ -AONT can be constructed with inputs of size  $k$ , secret output of length  $n$  and public output of length  $k$ , as follows:

$$T(x; r) = \langle r, f(r) \oplus x \rangle,$$

where  $r$  is a random string of length  $n$ ,  $r$  is the secret output, and  $f(r) \oplus x$  is the public output. Based on the authors' constructions of ERFs (which use strong extractors and pseudo-random generators), it follows that, whenever  $s = \ell^{\Theta(1)}$  and  $k = \ell^{O(1)}$ , there exists a computationally-secure  $\ell$ -AONT for messages of length  $k$  with secret output size  $s$  and public output size  $k$ .

Let us now analyze the applicability of the results of Canetti et al. to the various applications. The main concern is the difference made by the notion of splitting the

output into public and secret parts. This change to the model does not appear to affect the use of AONTs for the applications of efficient encryption and remotely-keyed encryption. The reason is that in those applications it is up to us to decide which part of the output to withhold from the adversary (see Figures 2-2 and 2-3). We can thus always decide to encrypt a piece of the secret output.

However, for the application of protection against exhaustive key search, the difference appears to be more substantial. Specifically, the adversary may be able to recover information about the input message by decrypting all of the secret part, but only a small piece (or even none) of the public part. Thus, the adversary's workload, as compared to encryption without an AONT, is only increased proportionally to the length of the secret part. The constructions of Canetti et al. would then appear useless for this application: The secret output length in those constructions is independent of the length of the message, and is exactly equal to the overhead (i.e., the expansion of the input introduced by the AONT). Thus, security can be increased only at the cost of increased overhead. On the other hand, the OAEP construction achieves security proportional to the length of the message, and independent of the overhead (since for OAEP, the whole output can be viewed as the secret part). It is interesting to note that if we were willing to have security proportional to the overhead, then we could achieve that using the following very simple construction, based on any cipher:

1. Generate  $s$  random keys  $k_1, \dots, k_s$ .
2. The secret output is  $k_1, \dots, k_s$ , and the public output is the encryption of the input message  $x$  using key  $k_1 \oplus \dots \oplus k_s$ .

Here  $s$  (number of blocks in the secret output) is the size of the overhead, as well as the increase in the adversary's workload. Clearly, if the adversary is missing any  $k_i$ , then she is unable to recover  $x$  (assuming that the underlying cipher is secure).

### Other Subsequent Work

Two subsequent papers analyze the security of AONTs in the context of applications. Desai [34] considers the application of protection against exhaustive key search. The

definition of AONTs used in that paper differs from ours in three important ways: First, the adversary’s view of the output is measured in blocks, and not in bits (similarly to Rivest’s [86] original definition). Second, the partial output (with at least one complete block removed) is required to not only convey no information about the input, but to also be indistinguishable from a random string. This strengthening appears necessary to achieve the required security results (see Section 2.3.1). Third, instead of specifying the positions of the missing bits (actually, blocks) in advance, the adversary is able to ask for blocks after seeing the value of other blocks (we will call this the *fully-adaptive* scenario). Desai then goes on to prove that encryption of AONT output does result in a secure encryption scheme (in the indistinguishability sense), and also slows down exhaustive search by a factor equal to the number of input blocks. Instead of the random oracle model which we use in this work, Desai’s model is based on the notion of an *ideal block cipher*, i.e., a function that gives an independent random permutation for any choice of the key. (The ideal block cipher model can be thought of as an analog of the random oracle model, except for block ciphers rather than hash functions. We note that the ideal block cipher model is used much less often in the literature than the random oracle model.) In that model, Desai presents an efficient construction of an AONT that is similar to Rivest’s construction, yet even simpler (the difference is that Desai’s construction uses  $h_i = m'_i$ , instead of  $h_i = E(K_0, m'_i \oplus i)$  — see the description of Rivest’s construction in Section 2.1.3). It would appear that Rivest’s construction could be proven secure in the ideal block cipher model as well.

Bellare and Boldyreva [4] analyze the security of AONTs (and in particular, OAEP) in the application of chaffing-and-winnowing. Their definition of AONT security is very similar to our non-adaptive indistinguishability model, except it is stated in terms of blocks (as in Rivest [86] and Desai [34]). The most relevant result of that paper is that the encryption scheme made by processing the message through an AONT and encrypting the first block is secure (the encryption could be done through chaffing-and-winnowing, or through any other scheme). Specifically, if the underlying encryption scheme is semantically secure, then the resulting scheme is semantically

secure. This proves the security of the use of AONTs for efficient encryption, as described at the beginning of Section 2.1.

### 2.1.5 Outline

The outline of the rest of this chapter is as follows. Section 2.2 describes the notation and model. In Section 2.3, we give formal definitions of security for AONTs and discuss the relation of our definitions to the applications. Section 2.4 presents the results on the security of OAEP as an AONT. Section 2.5 presents the proofs. Section 2.6 presents the conclusions and discusses open problems.

## 2.2 Notation and Model

Let us speak briefly about our notation and model. All algorithms used are oracle Turing machines, possibly randomized. Oracle queries execute in unit time. If  $A$  is a randomized algorithm, we may write  $A(x_1, \dots)$  to mean the distribution of  $A$ 's output on certain inputs. We may also specify the coins explicitly, as in  $A(r_A, x_1, \dots)$ , in which case the notation will refer to the fully determined output.

We will write  $x \stackrel{R}{\leftarrow} X$  to mean that a variable  $x$  is to be chosen at random according to distribution  $X$ . As a shorthand,  $x_1, x_2 \stackrel{R}{\leftarrow} X$  denotes  $x_1 \stackrel{R}{\leftarrow} X, x_2 \stackrel{R}{\leftarrow} X$ . On the other hand,  $x \leftarrow X$  will mean that  $x$  is to be set to the result of evaluating expression  $X$  (which is not random). If  $S$  is a set, then we will write  $x \stackrel{R}{\leftarrow} S$  to mean that  $x$  is chosen uniformly at random from  $S$ . We will write  $\Pr[x \stackrel{R}{\leftarrow} X; y \stackrel{R}{\leftarrow} Y; z \leftarrow Z; \dots : p(x, y, z, \dots)]$  to mean the probability of predicate  $p(x, y, z, \dots)$ , when  $x$  is chosen at random according to distribution  $X$ ,  $y$  is chosen at random according to distribution  $Y$ ,  $z$  is set to the result of evaluating expression  $Z$  (possibly a function of  $x$  and  $y$ ), etc. Similarly, we will write  $E[x \stackrel{R}{\leftarrow} X; \dots : f(x, \dots)]$  to mean the expected value of  $f(x, \dots)$  when  $x$  is chosen at random according to distribution  $X$ , etc.

To specify the distribution of a random function (“random oracle”), such as  $G$  and  $H$  for OAEP, we will use notation like  $G, H \stackrel{R}{\leftarrow} \Omega$ , where  $\Omega$  is the set of all maps from the set  $\{0, 1\}^*$  of finite strings to the set  $\{0, 1\}^\infty$  of infinite strings. The

notation should be interpreted as appropriate in its context, restricting the input and truncating the output of the function as necessary.

For a function  $f : X \rightarrow Y$ , we define its set-inverse  $f^{\{-1\}} : Y \rightarrow 2^X$  as  $f^{\{-1\}}(y) = \{x \in X : f(x) = y\}$ . We will freely put sets in positions that require a non-set, as in  $x \oplus Y$  or  $f(X, y)$  (where  $f$  might have signature  $\{0, 1\}^m \times \{0, 1\}^n \rightarrow \{0, 1\}^l$ ). This should be taken to mean the set of all possible results if all the elements of the specified set(s) are used in the specified position(s). For instance,

$$x \oplus Y = \{x \oplus y : y \in Y\}.$$

For  $x \in \{0, 1\}^*$ ,  $1 \leq i \leq |x|$ , and  $0 \leq l \leq |x| - i + 1$ , let  $\text{substr}(x, i, l)$  denote the substring of  $x$  starting at bit  $i$  (with the leftmost bit being 1) and having length  $l$ .

For any integer  $m$  and  $L \subseteq [1, m]$ , we define  $h_{m,L} : \{0, 1\}^m \rightarrow \{0, 1\}^{m-|L|}$  as follows:  $h_{m,L}$  takes a bit string of length  $m$  and throws out (“hides”) the bit positions indicated by  $L$ . More precisely, if we let  $\bar{L}_i$ , for  $1 \leq i \leq m - |L|$  denote the  $i$ th smallest element of  $\bar{L} = [1, m] \setminus L$ , then

$$h_{m,L}(x) = \text{substr}(x, \bar{L}_1, 1) \parallel \text{substr}(x, \bar{L}_2, 1) \parallel \cdots \parallel \text{substr}(x, \bar{L}_{m-|L|}, 1).$$

We also define  $u_{m,L} : \{0, 1\}^{|L|} \times \{0, 1\}^{m-|L|} \rightarrow \{0, 1\}^m$  as the inverse operation to  $h_{m,L}$ :  $u_{m,L}(v, x)$  returns the result of inserting (“unhiding”) the bits of  $v$  into  $x$  in the positions indicated by  $L$ . More precisely, we can define  $u_{m,L}$  as follows: for every  $1 \leq i \leq m$ ,

$$\text{substr}(u_{m,L}(v, x), i, 1) = \begin{cases} v_j & \exists j : i = L_j, \\ x_j & \exists j : i = \bar{L}_j, \end{cases}$$

where  $L_j$  denotes the  $j$ th smallest element of  $L$ . It is easy to see that for any  $x$  and

$v$ ,

$$\begin{aligned} h_{m,L}(u_{m,L}(v, x)) &= x, \\ h_{m,L}^{\{-1\}}(x) &= u_{m,L}(\{0, 1\}^{|L|}, x). \end{aligned}$$

For  $n' \geq l \geq 0$ , let  $\{n'_l\} = \{L \subseteq [1, n'] : |L| = l\}$ .

## 2.3 Definitions

Our definitions of security for AONTs are patterned after the notions of security for encryption defined in Goldwasser and Micali [46]: polynomial security (polynomial indistinguishability) and semantic security.<sup>1</sup> We also try to define security “exactly,” as in Bellare and Rogaway [10]: instead of concerning ourselves with asymptotics (i.e., showing that the adversary’s advantage is “negligible” in the security parameters), we are interested in giving an exact bound on the adversary’s advantage, as a function of the adversary’s running time, the number of bits of AONT’s output given to the adversary, etc.

For simplicity, we will formulate the definitions in terms of a single random oracle  $\Gamma$ . No generality is lost, since a single random oracle can be used to simulate several, by constructing the query as the concatenation of the oracle index and the original query. For instance, we could use  $\Gamma$  to simulate random oracles  $G$  and  $H$  by translating query  $x$  to  $G$  into query  $0\|x$  to  $\Gamma$  and query  $y$  to  $H$  into  $1\|y$ . In addition, it would be easy to change the definitions for the case of no random oracles.

The *non-adaptive indistinguishability scenario* is as follows: Let  $L$  be an arbitrary set of  $l$  bit positions. The adversary runs in two stages:

1. **Find stage:** The adversary is given  $L$  and access to  $\Gamma$ . She outputs  $x_0 \in \{0, 1\}^n$ ,  $x_1 \in \{0, 1\}^n$ , and  $c_f \in \{0, 1\}^*$ .

---

<sup>1</sup>To prevent confusion, we note that while Bellare and Rogaway [10] talk about semantic security (for encryption), the definition they give is actually stated in terms of indistinguishability. This is acceptable in their context, since the two notions are known to be equivalent for encryption (see Micali et al. [72]). In our context, however, we state and analyze each one separately, since no equivalence has yet been proven.

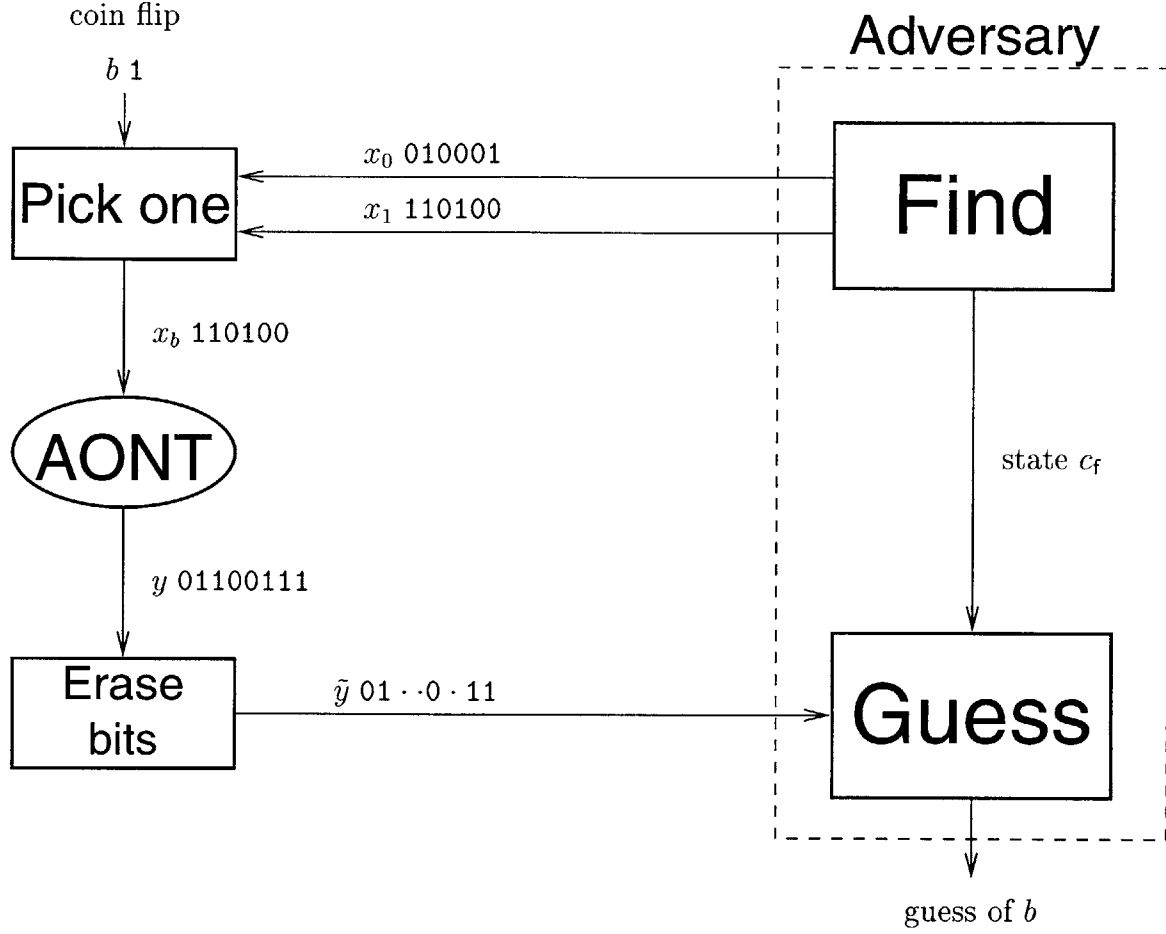


Figure 2-5: Diagram of the non-adaptive indistinguishability scenario. Here  $\tilde{y} = h_{n',L}(y)$ . In this figure and below, the bit strings are shown for illustration purposes only.

2. **Guess stage:** The adversary is given  $c_f$  and, for random bit  $b$ ,  $\text{AONT}^\Gamma(x_b)$  with bit positions  $L$  missing. The adversary has access to  $\Gamma$ . She has to guess  $b$ .

We want the adversary's probability of correctly guessing  $b$  to be as close as possible to  $\frac{1}{2}$ . Note that  $x_0$  and  $x_1$  do not need to be explicitly passed to the guess stage, since they may be included in  $c_f$ . We may view  $c_f$  as the saved state of the adversary at the end of the find stage.

A diagram of the non-adaptive indistinguishability scenario is shown in Figure 2-5. The formal definition is as follows:

**Definition 1 (Non-adaptive indistinguishability).** Let AONT be a randomized transform mapping  $n$ -bit messages to  $n'$ -bit outputs and using random oracle  $\Gamma$ . Let  $l$  be between 1 and  $n'$ . An adversary  $A$  is said to succeed in  $(T, q_\Gamma, \epsilon)$ -**distinguishing** AONT with  $l$  missing bits if there exists  $L \in \{n'_l\}$  such that

$$\Pr[\Gamma \xleftarrow{R} \Omega; (x_0, x_1, c_f) \xleftarrow{R} A^\Gamma(L, \text{find}); b \xleftarrow{R} \{0, 1\}; \\ y \xleftarrow{R} \text{AONT}^\Gamma(x_b) : A^\Gamma(h_{n',L}(y), c_f, \text{guess}) = b] \geq \frac{1}{2} + \epsilon,$$

and, moreover, in the experiment above,  $A$  runs for at most  $T$  steps, and makes at most  $q_\Gamma$  queries to  $\Gamma$ .

It follows from this definition that in order for an AONT to be secure in the sense of non-adaptive indistinguishability for certain choices of parameters, it needs to be that for every adversary and every  $L$ , the adversary's advantage has to be less than  $\epsilon$ .

The *adaptive indistinguishability scenario* is as follows: The adversary runs in three stages. The first stage chooses a value of  $L$ , while the last two stages are same as in the non-adaptive indistinguishability scenario. The adversary runs as follows:

1. **Select stage:** The adversary is given  $l$  and access to  $\Gamma$ . She selects  $l$  bit positions and outputs  $L \in \{n'_l\}$  and  $c_s \in \{0, 1\}^*$ .
2. **Find stage:** The adversary is given  $c_s$  and access to  $\Gamma$ . She outputs  $x_0 \in \{0, 1\}^n$ ,  $x_1 \in \{0, 1\}^n$ , and  $c_f \in \{0, 1\}^*$ .
3. **Guess stage:** The adversary is given  $c_f$  and, for random bit  $b$ ,  $\text{AONT}^\Gamma(x_b)$  with bit positions  $L$  missing. The adversary has access to  $\Gamma$ . She has to guess  $b$ .

Similarly to the remark about  $x_0$  and  $x_1$  above, we note that  $L$  does not need to be explicitly passed to the find and guess stages, since it may be included in  $c_s$ , and then put into  $c_f$ .

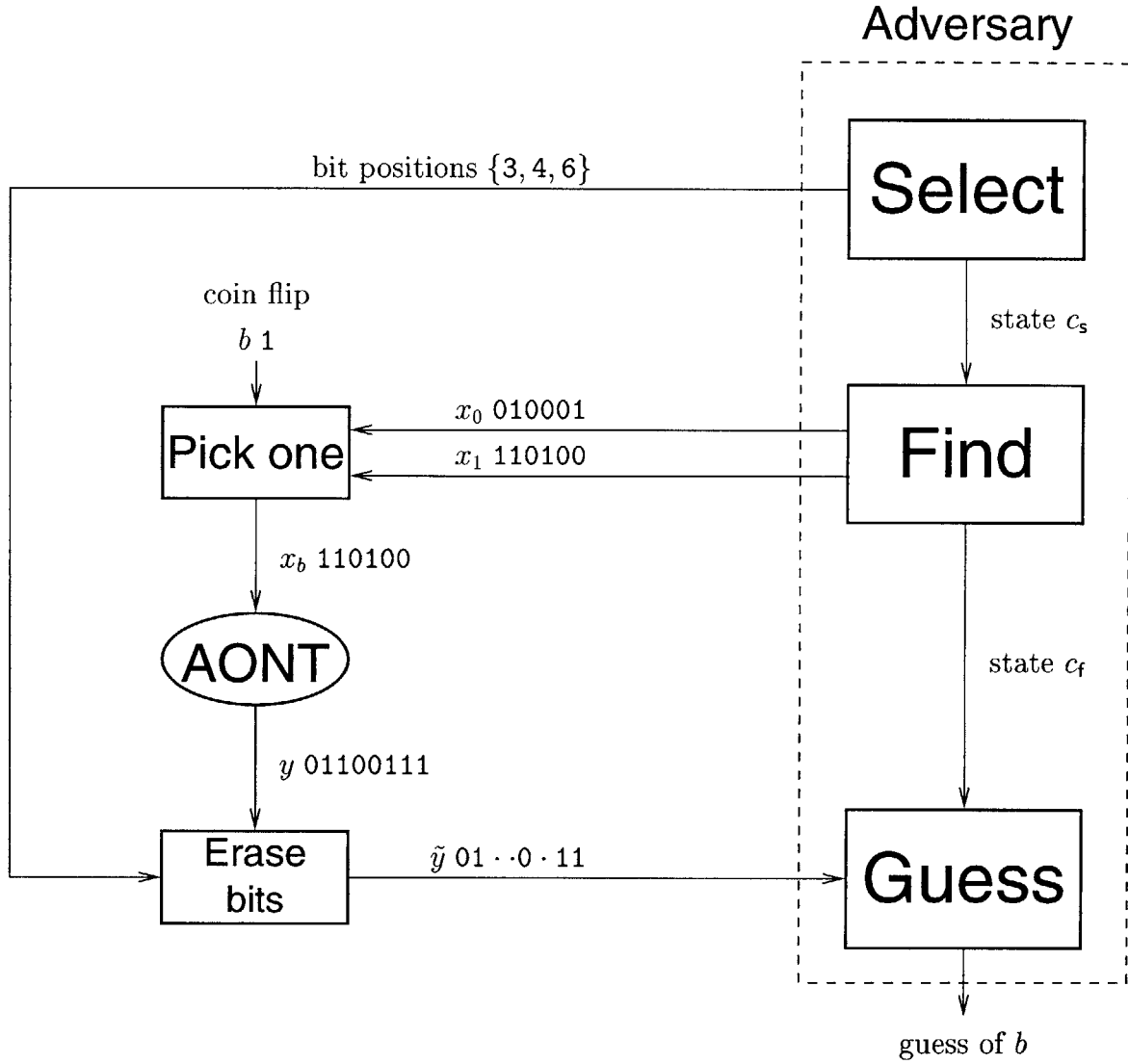


Figure 2-6: Diagram of the adaptive indistinguishability scenario. Here  $\tilde{y} = h_{n',L}(y)$ .

A diagram of the adaptive indistinguishability scenario is shown in Figure 2-6. In the formal definition, we will assume that the adversary's select stage will always output a valid value of  $L \in \{l_i^{n'}\}$  (this can be implemented by having a suitable encoding).

**Definition 2 (Adaptive indistinguishability).** *Let AONT be a randomized transform mapping  $n$ -bit messages to  $n'$ -bit outputs and using random oracle  $\Gamma$ . Let  $l$  be between 1 and  $n'$ . An adversary  $A$  is said to succeed in  $(T, q_\Gamma, \epsilon)$ -**adaptively-distinguishing** AONT with  $l$  missing bits if*

$$\Pr[\Gamma \stackrel{R}{\leftarrow} \Omega; (L, c_s) \stackrel{R}{\leftarrow} A^\Gamma(l, \text{select}); (x_0, x_1, c_f) \stackrel{R}{\leftarrow} A^\Gamma(c_s, \text{find}); \\ b \stackrel{R}{\leftarrow} \{0, 1\}; y \stackrel{R}{\leftarrow} \text{AONT}^\Gamma(x_b) : A^\Gamma(h_{n',L}(y), c_f, \text{guess}) = b] \geq \frac{1}{2} + \epsilon,$$

and, moreover, in the experiment above,  $A$  runs for at most  $T$  steps, and makes at most  $q_\Gamma$  queries to  $\Gamma$ .

Note that for the application of speeding up encryption that was mentioned above in Section 2.1, it is sufficient for the AONT to be secure for a fixed choice of the missing part of the output (since the user decides which part will be encrypted). Thus, for that application, it is sufficient for the AONT to be secure in the non-adaptive scenario. However, when an AONT is used to increase the cost of exhaustive search, it needs to be secure in the adaptive scenario, since then the adversary has a choice of which blocks to decrypt.

For the adaptive and non-adaptive indistinguishability scenarios, we will assume, without loss of generality, that  $A$  never asks the same oracle query more than once ( $A$  can accomplish this by remembering the history of past queries; this history can be passed between stages through  $c_s$  and  $c_f$ ).

The *non-adaptive semantic security scenario* is as follows: Let  $L$  be an arbitrary set of  $l$  bit positions and  $f : \{0, 1\}^n \rightarrow \{0, 1\}^*$  be an arbitrary deterministic function. The adversary runs in two unconnected stages (each stage can be viewed as a separate algorithm):

- **Find stage:** The adversary is given  $L$  and access to  $\Gamma$ . She outputs  $x \in \{0, 1\}^n$ .
- **Guess stage** (no data is passed from the find stage): The adversary is given  $L$  and  $\text{AONT}^\Gamma(x)$  with bit positions  $L$  missing. The adversary has access to  $\Gamma$ . She has to guess  $f(x)$ .

In the context of the traditional definition of semantic security for encryption, the adversary's find stage may be seen as the sampling algorithm for a distribution of messages, and the guess stage as the actual predicting algorithm. We want the adversary not to be able to do substantially better than always outputting the most probable value of  $f(x)$ .

A diagram of the non-adaptive indistinguishability scenario is shown in Figure 2-7. The formal definition is as follows:

**Definition 3 (Non-adaptive semantic security).** *Let  $\text{AONT}$  be a randomized transform mapping  $n$ -bit messages to  $n'$ -bit outputs and using random oracle  $\Gamma$ . Let  $l$  be between 1 and  $n'$ . Let  $f : \{0, 1\}^n \rightarrow \{0, 1\}^*$  be any deterministic function. An adversary  $A$  is said to succeed in  $(T, q_\Gamma, \epsilon)$ -**predicting**  $f$  from  $\text{AONT}$  with  $l$  missing bits if there exists  $L \in \binom{[n']}{l}$  such that*

$$\Pr[\Gamma \stackrel{R}{\leftarrow} \Omega; x \stackrel{R}{\leftarrow} A^\Gamma(L, \text{find}); y \stackrel{R}{\leftarrow} \text{AONT}^\Gamma(x) : A^\Gamma(L, h_{n',L}(y), \text{guess}) = f(x)] \geq p_{A,f} + \epsilon, \quad (2.1)$$

where

$$p_{A,f} = E[\Gamma \stackrel{R}{\leftarrow} \Omega : \max_z \Pr[x \stackrel{R}{\leftarrow} A^\Gamma(L, \text{find}) : f(x) = z]],$$

and, moreover, in the experiment (2.1),  $A$  runs for at most  $T$  steps, and makes at most  $q_\Gamma$  queries to  $\Gamma$ .

The expectation in the definition of  $p_{A,f}$  is necessary to handle the possibility that the adversary may choose  $x$  to be a function of  $\Gamma$  (e.g.,  $x$  could be set to the result of querying  $\Gamma$  on some fixed input). This would result in perfect prediction (both the

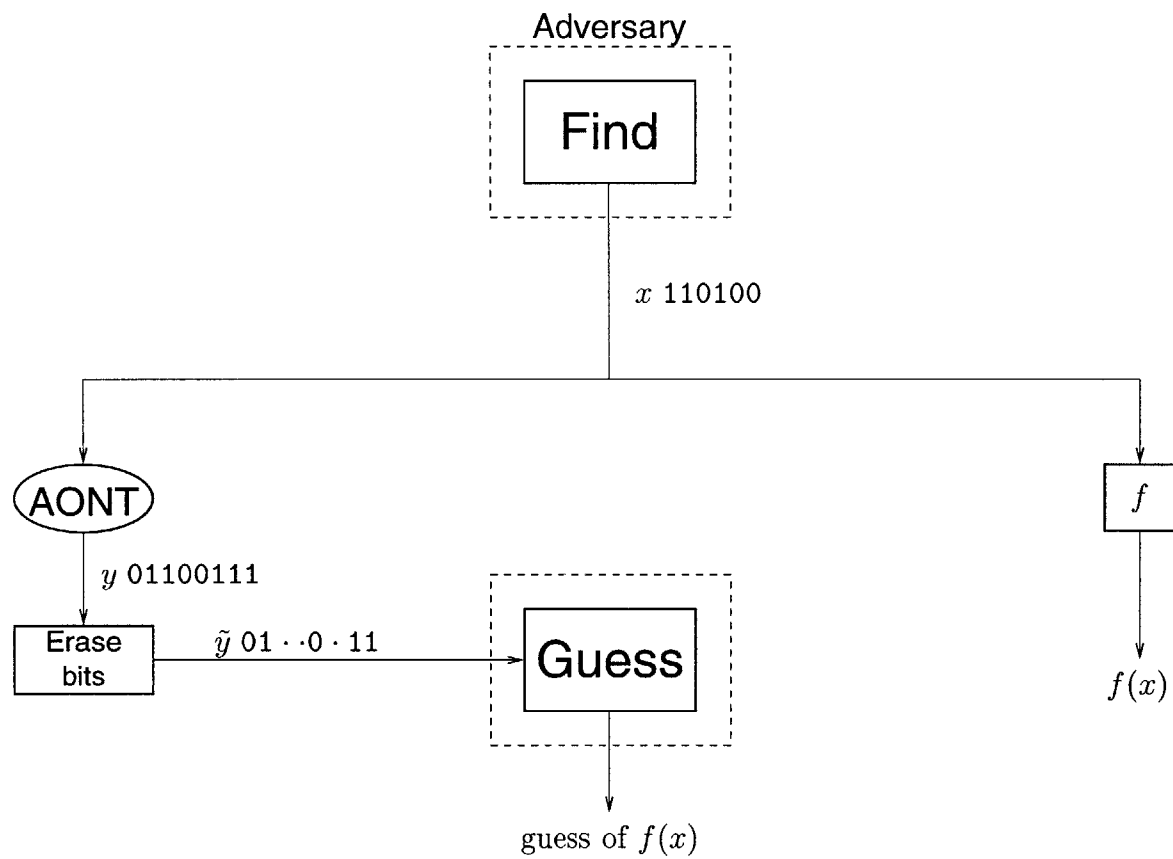


Figure 2-7: Diagram of the non-adaptive semantic security scenario. Here  $\tilde{y} = h_{n',L}(y)$ . Note that no state information is passed between the adversary's stages.

find and guess stages can compute the same  $x$ ), even though the output of the find stage will appear random, for random  $\Gamma$ . Thus, the quantity

$$\max_z \Pr[\Gamma \xleftarrow{R} \Omega; x \xleftarrow{R} A^\Gamma(L, \text{find}) : f(x) = z]$$

could be much smaller than the adversary's success probability. However, for any fixed  $\Gamma$ , this adversary would always output the same  $x$ , so  $p_{A,f} = 1$ . Thus, this adversary's advantage  $\epsilon$  will have to be zero.

In the semantic security scenario (both adaptive and non-adaptive), no information is passed between the adversary's find and guess stages, except  $h_{n',L}(\text{AONT}^\Gamma(x))$  (otherwise, the find stage could simply pass the value of  $f(x)$ ). We will therefore remove the assumption that  $A$  can't make the same query to  $\Gamma$  more than once. We will still assume, though, that all queries are unique within a single stage.

The *adaptive semantic security scenario* is same as the non-adaptive one, except for the addition of the select stage before the find stage, in which the adversary outputs  $L$ . The formal definition is as follows:

**Definition 4 (Adaptive semantic security).** *Let AONT be a randomized transform mapping  $n$ -bit messages to  $n'$ -bit outputs and using random oracle  $\Gamma$ . Let  $l$  be between 1 and  $n'$ . Let  $f : \{0,1\}^n \rightarrow \{0,1\}^*$  be any deterministic function. An adversary  $A$  is said to succeed in  $(T, q_\Gamma, \epsilon)$ -**adaptively-predicting**  $f$  from AONT with  $l$  missing bits if*

$$\Pr[\Gamma \xleftarrow{R} \Omega; (L, c_s) \xleftarrow{R} A^\Gamma(l, \text{select}); x \xleftarrow{R} A^\Gamma(c_s, \text{find}); y \xleftarrow{R} \text{AONT}^\Gamma(x) : A^\Gamma(h_{n',L}(y), c_s, \text{guess}) = f(x)] \geq p_{A,f} + \epsilon, \quad (2.2)$$

where

$$p_{A,f} = E[\Gamma \xleftarrow{R} \Omega; (L, c_s) \xleftarrow{R} A^\Gamma(l, \text{select}) : \max_z \Pr[x \xleftarrow{R} A^\Gamma(c_s, \text{find}) : f(x) = z]],$$

and, moreover, in the experiment (2.2),  $A$  runs for at most  $T$  steps, and makes at

most  $q_\Gamma$  queries to  $\Gamma$ .

Note that since information may be passed from the select stage to the find and guess stages (through  $c_s$ ), we can assume that no query from the select stage is repeated in any of the other stages. There is no danger in passing  $c_s$  to the guess stage, since  $c_s$  is generated before  $x$  is chosen (note that  $p_{A,f}$  involves an expectation over  $c_s$ , so the adversary will not gain any advantage by choosing  $(x, f(x))$  at the select stage and then passing it to the other stages).

### 2.3.1 Relation of the Definitions to the Applications

Once the definitions have been formulated, it is appropriate to ask whether or not they are necessary and sufficient for the applications described above. It is clear from the motivation given in Section 2.1 that non-adaptive security, in terms of either indistinguishability or semantic security, appears to be necessary for all the applications mentioned above: protection against exhaustive key search, as well as encryption. As mentioned above in Section 2.3, we actually need adaptive security for the first application, since it is up to the adversary to choose which blocks to decrypt.

However, we don't necessarily need the bit positions to be arbitrary. In fact, for both of those applications it appears sufficient to have granularity at the block, rather than the bit level (similarly to Rivest's [86] original definition). Our main motivation for considering arbitrary sets of bit positions was because we wanted to show the strongest possible results for OAEP.

We note that the application of *protection against partial key exposure*, proposed by the subsequent paper of Canetti et al. [23], provides some extra motivation for considering bit-level granularity. The scenario in that application is that the adversary finds out part of a cryptographic key. We would like to maintain some security guarantees, based on the amount of information leaked. However, most cryptographic schemes do not offer any guarantees of security if even a small piece of the secret is published. The solution is to store the key processed with an AONT, in which case a partial leakage will not give any information to the adversary. To analyze security of

AONTs in this context, it is appropriate to have the maximum possible granularity, since that would allow the maximum potential for spreading the secret information and making it hard to find for the adversary. In addition to this application, Canetti et al. mention the possibility of viewing an AONT as a secret-sharing scheme with bit-sized shares, which also motivates analyzing security at that level of granularity.

Some results on the sufficiency of our definitions are given in the papers mentioned in Section 2.1.4. One of the results of Bellare and Boldyreva [4] can be interpreted as showing that our non-adaptive definition is sufficient for the semantic security of the encryption scheme based on AONTs, as long as the underlying encryption algorithm is semantically secure. This result also implies security in the context of remotely-keyed encryption, as long as the adversary has no access to the internals of the encryption mechanism (i.e., the adversary is not allowed to perform such operations as impersonating the smartcard). There is still an open question as to what kind of security is achieved against stronger attack scenarios, both for general encryption (e.g., chosen-ciphertext attacks), as well as for remotely-keyed encryption (e.g., security against forging valid plaintext/ciphertext pairs [58]).

The results of Desai [34] appear to imply that our adaptive definition implies protection against exhaustive key search, as long as two additional requirements are made: First, partial AONT output should not only convey no information about the input, but also be indistinguishable from a random string. The extra requirement does appear to be necessary, as illustrated by the following example: Suppose we append 0 to the output of an AONT. The resulting scheme is still a secure AONT (with security just slightly decreased), and yet the adversary can reject half the keys in an exhaustive search by simply checking the block that is supposed to always end with 0. The second requirement is that instead of specifying the positions of the missing bits (or blocks) in advance, the adversary should be able to ask for blocks after seeing the value of other blocks (we call this the *fully-adaptive* scenario). We have not analyzed the security of OAEP in the fully-adaptive scenario, and it is indeed an important open problem.

## 2.4 Security Results

Throughout most of this section we will be using two random oracles  $G$  and  $H$ . As mentioned above, we can still use our definitions, since  $\Gamma$  could be used to simulate  $G$  and  $H$ . We will write  $A^{G,H}$  in place of  $A^\Gamma$ . We will also use notation  $(T, q_G, q_H, \epsilon) \cdots$  (e.g., “an adversary  $(T, q_G, q_H, \epsilon)$ -distinguishes”) as a shorthand for  $(T, q_G + q_H, \epsilon) \cdots$ , with the additional condition that at most  $q_G$  queries are made to  $G$  and at most  $q_H$  queries are made to  $H$ .

### 2.4.1 Non-adaptive Indistinguishability: Upper Bound

**Theorem 1.** *Suppose  $l \leq k_0$  and  $k_0 \geq 14$ . Suppose that there exists an adversary  $A$  that  $(T, q_G, q_H, \epsilon)$ -distinguishes OAEP with  $l$  missing bits, where  $q_G \leq 2^{k_0-1}$ . Then*

$$\epsilon \leq 8q_G \frac{k_0}{\log_2 k_0} 2^{-l}.$$

The proof appears in Section 2.5.1. The intuition behind the result is as follows: Let  $r_0$  be the value of  $r$  that was used to generate  $\tilde{y} = h_{n',L}(\text{OAEP}^{G,H}(x_b))$  in a particular experiment. Then, the adversary cannot find out any information about  $x_b$  unless she queries  $G$  for  $G(r_0)$  (since  $x_b$  only appears in  $\text{OAEP}^{G,H}(x_b, r_0)$  as  $x_b \oplus G(r_0)$ ). There are  $\sim 2^l$  possible values of  $r_0$ , corresponding to the  $2^l$  values of  $y$  that are consistent with  $\tilde{y}$ . Thus we would expect the probability that any of the adversary’s queries to  $G$  are equal to  $r_0$  to be bounded by approximately  $q_G 2^{-l}$ . The complication is that there may be fewer than  $2^l$  possible values of  $r_0$  and that these values may not be equally probable, given  $\tilde{y}$ . These possible variations in probability cause the term  $O(\frac{k_0}{\log k_0})$ .

Note that this result, like all the others in this chapter, does not use any computational assumptions and the bound is information theoretic, based on the properties of random oracles. In fact, the bound does not directly depend on  $T$ , the adversary’s running time. It does, however, have implications for the running time, since  $T \geq q_G + q_H$  (every oracle query takes unit time).

### 2.4.2 Non-adaptive Indistinguishability: Lower Bound

To see how good our upper bound is, let us try to give a lower bound on the adversary's advantage, by estimating the success of exhaustive search. This lower bound applies to any AONT.

**Theorem 2.** *Let AONT be a randomized transform mapping  $n$ -bit messages to  $n'$ -bit outputs and using random oracle  $\Gamma$ . Let  $l$  be between 1 and  $n - 3$ . Then, for any  $L \in \{l^{n'}\}$  and any  $N$  between 1 and  $2^l$ , there exists an adversary that  $(NT, Nq_\Gamma, \epsilon)$ -distinguishes AONT with  $l$  missing bits, with*

$$\epsilon \geq \frac{1}{16} N 2^{-l}.$$

Here  $T$  and  $q_\Gamma$  are the time and number of queries to  $\Gamma$ , respectively, taken by a single evaluation of AONT.

The proof outline appears in Section 2.5.2. The idea of the proof is as follows: The exhaustive search algorithm that achieves the advantage of at least  $\frac{1}{16} N 2^{-l}$  works by choosing  $x_0$  and  $x_1$  independently at random in the find stage. The guess stage tries random values of the missing bits, up to  $N$  times, and, if the inverse AONT returns  $x_{b'}$  for  $b' \in \{0, 1\}$ , produces  $b'$  as the guess. If none of the trials has succeeded, a random bit is returned. The idea of the analysis of this algorithm is that every trial in the guess stage has probability of at least  $2^{-l}$  of succeeding with the correct value of  $b$  (since there exists a choice of the missing bits, namely the values that actually appeared in  $y$ , that leads to  $x_b$ ). On the other hand, since  $x_{1-b}$  is chosen uniformly and independently, the probability of getting  $x_{1-b}$  in any particular trial is  $2^{-n} \leq 2^{-l-3}$ .

We see from Theorems 1 and 2, that no adversary can improve by a factor of more than  $O(\frac{k_0}{\log k_0})$  over exhaustive search. Since, for large  $l$ , this factor is negligible compared to  $2^{-l}$ , our bounds for OAEP are nearly optimal.

We also see that no AONT can be substantially more secure than OAEP, in the sense that no AONT can have an upper bound that is better than OAEP's by a factor of more than  $O(\frac{k_0}{\log k_0})$ .

### 2.4.3 Adaptive Indistinguishability

**Theorem 3.** *Suppose  $l \leq k_0$ ,  $l \leq \frac{n}{2}$ , and  $k_0 \geq 14$ . Suppose that there exists an adversary  $A$  that  $(T, q_G, q_H, \epsilon)$ -adaptively-distinguishes OAEP with  $l$  missing bits, where  $q_G \leq 2^{k_0-1}$ . Then*

$$\epsilon \leq 8(q_G + q_H) \frac{k_0}{\log_2 k_0} 2^{-l}.$$

The proof outline appears in Section 2.5.3. It is very similar to the proof of Theorem 1, with the only major difference being that we have to consider the possible correlation between  $L$  and  $H$  (since  $L$  may be chosen by the adversary to depend on  $H$ ). This is taken care of by showing that with large probability (depending on  $q_H$ ), the queries made in the select stage will not constrain  $H$  enough to spoil those properties of it that are used in the proof of Theorem 1.

We can easily see that for the adaptive indistinguishability scenario, as for the non-adaptive one, our bound is optimal within a factor of  $O(\frac{k_0}{\log k_0})$  of the advantage given by exhaustive search for an arbitrary AONT (in this scenario, exhaustive search would choose a random  $L$  in the select stage).

### 2.4.4 Non-adaptive Semantic Security

**Theorem 4.** *Suppose  $l \leq k_0$  and  $k_0 \geq 14$ . Suppose that there exists a deterministic function  $f : \{0, 1\}^n \rightarrow \{0, 1\}^*$  and an adversary  $A$  that  $(T, q_G, q_H, \epsilon)$ -predicts  $f$  from OAEP with  $l$  missing bits, where  $q_G \leq 2^{k_0-1}$ . Then*

$$\epsilon \leq 8q_G \frac{k_0}{\log_2 k_0} 2^{-l}.$$

The proof outline appears in Section 2.5.4. It is a simple modification of the proof of Theorem 1, with the difference being the estimation of the adversary's success probability in the case where she has not queried  $G$  for  $G(r_0)$  (where  $r_0$  is the value of  $r$  used to compute  $\text{OAEP}^{G,H}(x)$  in the experiment). In the case of Theorem 1 that success probability was  $\frac{1}{2}$ , while here it can be easily seen to be less than or equal to

$p_{A,f}$ .

We have not yet shown a lower bound for the semantic security scenarios. We still expect our upper bound for these scenarios to be nearly optimal, as for the indistinguishability scenarios.

## 2.4.5 Adaptive Semantic Security

**Theorem 5.** *Suppose  $l \leq k_0$ ,  $l \leq \frac{n}{2}$ , and  $k_0 \geq 14$ . Suppose that there exists a deterministic function  $f : \{0, 1\}^n \rightarrow \{0, 1\}^*$  and an adversary  $A$  that  $(T, q_G, q_H, \epsilon)$ -adaptively-predicts  $f$  from OAEP with  $l$  missing bits, where  $q_G \leq 2^{k_0-1}$ . Then*

$$\epsilon \leq 8(q_G + q_H) \frac{k_0}{\log_2 k_0} 2^{-l}.$$

The proof of Theorem 5 may be easily obtained from the proof of Theorem 3 by adding the modifications given in the outline for the proof of Theorem 4.

## 2.5 Proofs of Theorems

### 2.5.1 Proof of Theorem 1

Let  $A$  be an adversary. We want to show that

$$\Pr[G, H \xleftarrow{R} \Omega; (x_0, x_1, c_f) \xleftarrow{R} A^{G,H}(L, \text{find}); b \xleftarrow{R} \{0, 1\}; r_0 \xleftarrow{R} \{0, 1\}^{k_0}; \\ y \leftarrow \text{OAEP}^{G,H}(x_b, r_0) : A^{G,H}(h_{n',L}(y), c_f, \text{guess}) = b] \leq \frac{1}{2} + 8q_G \frac{k_0}{\log_2 k_0} 2^{-l}$$

for all  $L \in \{l^{n'}\}$  and  $k_0 \geq 14$  (recall that for OAEP we have  $n' = n + k_0$ ). Note that we have explicitly mentioned random variable  $r_0$ , which supplies the randomness for the computation of  $\text{OAEP}^{G,H}(x_b)$ . We will refer to the success of the adversary in the above experiment as event AC (Adversary is Correct). In the rest of the proof we will refer freely to random variables from the experiment.

Let  $q_{FG}$  and  $q_{GG} = q_G - q_{FG}$  be the number of queries that  $A$  makes to  $G$  in

the find and guess stages respectively (without loss of generality, we can assume that these are fixed for any given values of  $n$  and  $k_0$ ).

Fix a value of  $L \in \{l^{n'}\}$ . Let  $L_s = L \cap [1, n]$  and  $L_t = (L \cap [n+1, n+k_0]) - n$ . We can think of  $L_s$  and  $L_t$  as the sets of bits that are removed in the  $s$  and  $t$  parts of  $y$ , respectively. Let  $l_s = |L_s|$  and  $l_t = |L_t|$ . Clearly,  $l = l_s + l_t$ .

Let  $\tilde{y}$  denote the random variable  $h_{n',L}(y)$ . Let

$$\begin{aligned}\tilde{s} &= \text{substr}(\tilde{y}, 1, n - l_s), \\ \tilde{t} &= \text{substr}(\tilde{y}, n - l_s + 1, k_0 - l_t).\end{aligned}$$

Thus  $\tilde{s}$  and  $\tilde{t}$  correspond to the  $s$  and  $t$  parts of  $y$  with bits from  $L$  removed, just as  $\tilde{y}$  corresponds to  $y$  with bits from  $L$  removed.

Let  $h_s : \{0, 1\}^n \rightarrow \{0, 1\}^{n-l_s}$  and  $h_t : \{0, 1\}^{k_0} \rightarrow \{0, 1\}^{k_0-l_t}$  be defined as  $h_{n,L_s}$  and  $h_{k_0,L_t}$ , respectively. We may look at  $h_s$  and  $h_t$  as functions that remove the missing bits from the  $s$  and  $t$  parts of  $y$ , respectively. Let  $u_s : \{0, 1\}^{l_s} \times \{0, 1\}^{n-l_s} \rightarrow \{0, 1\}^n$  and  $u_t : \{0, 1\}^{l_t} \times \{0, 1\}^{k_0-l_t} \rightarrow \{0, 1\}^{k_0}$  be defined as  $u_{n,L_s}$  and  $u_{k_0,L_t}$ . We may look at  $u_s$  and  $u_t$  as functions that “fill in the holes” in the  $s$  and  $t$  parts of  $y$ , respectively.

Note that the subscripts in  $L_s$ ,  $L_t$ ,  $l_s$ ,  $l_t$ ,  $h_s$ ,  $h_t$ ,  $u_s$ , and  $u_t$  refer to the part of OAEP output that the object (set, number, of function) is related to. They do not imply dependence on the actual values of  $s$  and  $t$ , taken as substrings of the random variable  $y$  (hence the subscripts are set in Roman font, and not in italics).

Fig. 2-8 shows the relationship between the above mentioned variables and functions.

Let

$$\begin{aligned}\mathcal{S} &= h_s^{\{-1\}}(\tilde{s}) = u_s(\{0, 1\}^{l_s}, \tilde{s}), \\ \mathcal{T} &= h_t^{\{-1\}}(\tilde{t}) = u_t(\{0, 1\}^{l_t}, \tilde{t}).\end{aligned}$$

We can think of  $\mathcal{S}$  and  $\mathcal{T}$  as the sets of the possible values of  $s$  and  $t$ , respectively,

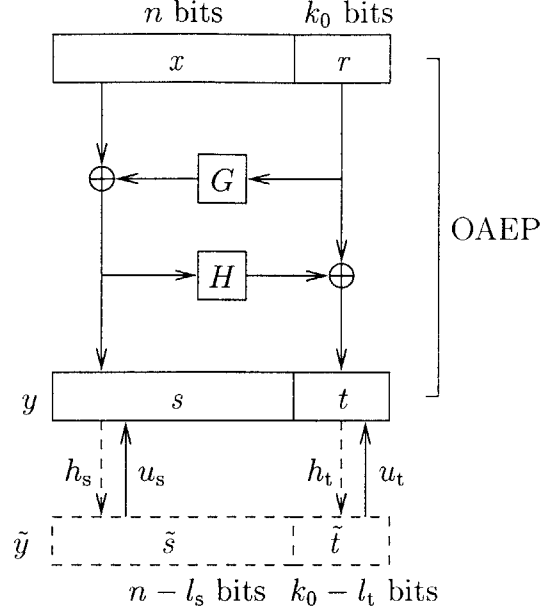


Figure 2-8: OAEP and related functions

that are consistent with the information in  $\tilde{y}$ . Let

$$\begin{aligned}\mathcal{S}_r^H &= \{s \in \mathcal{S} : r \oplus H(s) \in \mathcal{T}\} \quad \text{for } r \in \{0, 1\}^{k_0}, \\ \mathcal{R}^{G,H} &= \{r \in \{0, 1\}^{k_0} : x_b \oplus G(r) \in \mathcal{S}_r^H\}.\end{aligned}$$

$\mathcal{S}_r^H$  may be seen as the set of values of  $s$  that are consistent with  $\tilde{y}$  and a particular  $r$  for some  $G$ :

$$\begin{aligned}\mathcal{S}_r^H &= \{s \in \{0, 1\}^n : \exists G \in \Omega : \text{substr}(\text{OAEP}^{G,H}(x_b, r), 1, n) = s \wedge \\ &\quad h_{n',L}(\text{OAEP}^{G,H}(x_b, r)) = \tilde{y}\}.\end{aligned}$$

$\mathcal{R}^{G,H}$  may be seen as the set of values of  $r$  that are consistent with  $x_b$  and  $\tilde{y}$ . In other words, it is the set of possible values of  $r_0$  that could have been used to produce the adversary's view for the value of  $b$  that was used:

$$\mathcal{R}^{G,H} = \{r \in \{0, 1\}^{k_0} : h_{n',L}(\text{OAEP}^{G,H}(x_b, r)) = \tilde{y}\}.$$

Define  $H' : \{0, 1\}^{l_s} \rightarrow \{0, 1\}^{k_0 - l_t}$  as

$$H'(v) = h_t(H(u_s(v, \tilde{s}))).$$

Note that  $H'$  does not depend on any values of  $H$ , except within domain  $\mathcal{S}$ . If  $H$  is a random function, then, for any fixed  $\tilde{s}$ ,  $H'$  is also a random function. Now  $\mathcal{S}_r^H = \{u_s(v, \tilde{s}) : v \in V_{h_t(r)}^{H'}\}$ , where  $V_{r'}^{H'} = \{v \in \{0, 1\}^{l_s} : r' \oplus H'(v) = \tilde{t}\}$ . In particular,  $|\mathcal{S}_r^H| = |V_{h_t(r)}^{H'}| = |H'^{\{-1\}}(h_t(r) \oplus \tilde{t})|$ .

Let  $M_{H'} = \max_{z \in \{0, 1\}^{k_0 - l_t}} |H'^{\{-1\}}(z)|$ .

Let  $\text{AskR}_0$  be the event that  $A$  ever asks for the value of  $G(r_0)$ . Let  $\text{FAskR}_0$  and  $\text{GAskR}_0$  be the events that such a query is made during the find and the guess stages, respectively. Note that  $\text{FAskR}_0$  and  $\text{GAskR}_0$  are mutually exclusive, since, by assumption, all of  $A$ 's oracle queries are unique. We have

$$\begin{aligned} \Pr[\text{AC}] &= \Pr[\text{AC} | \neg \text{AskR}_0] \cdot \Pr[\neg \text{AskR}_0] + \Pr[\text{AC} | \text{AskR}_0] \cdot \Pr[\text{AskR}_0] \\ &\leq \Pr[\text{AC} | \neg \text{AskR}_0] + \Pr[\text{AskR}_0] \\ &= \Pr[\text{AC} | \neg \text{AskR}_0] + \Pr[\text{FAskR}_0] + \Pr[\text{GAskR}_0] \\ &= \Pr[\text{AC} | \neg \text{AskR}_0] + \Pr[\text{FAskR}_0] + \Pr[\text{GAskR}_0 \wedge \neg \text{FAskR}_0] \\ &= \Pr[\text{AC} | \neg \text{AskR}_0] + \Pr[\text{FAskR}_0] + \Pr[\neg \text{FAskR}_0] \cdot \Pr[\text{GAskR}_0 | \neg \text{FAskR}_0] \\ &\leq \Pr[\text{AC} | \neg \text{AskR}_0] + \Pr[\text{FAskR}_0] + \Pr[\text{GAskR}_0 | \neg \text{FAskR}_0]. \end{aligned}$$

We have

$$\Pr[\text{AC} | \neg \text{AskR}_0] = \frac{1}{2},$$

since if  $A$  does not see  $G(r_0)$ , all of its inputs will be independent of  $b$  ( $x_b$  only appears in  $y$  as  $x_b \oplus G(r_0)$ ). We also have

$$\Pr[\text{FAskR}_0] \leq q_{\text{FG}} 2^{-k_0},$$

since  $r_0$  is chosen at random after the find stage.

Let's bound  $\Pr[\text{GAskR}_0 | \neg \text{FAskR}_0]$ . Let  $\text{GAskR}_0^i$  denote the event that  $A$ 's  $i$ th query to  $G$  in the guess stage is  $r_0$ . The  $\text{GAskR}_0^i$ 's are all mutually exclusive. Therefore,

$$\begin{aligned}
& \Pr[\text{GAskR}_0 | \neg \text{FAskR}_0] \\
&= \Pr\left[\bigvee_{i=1}^{q_{\text{GG}}} \text{GAskR}_0^i \mid \neg \text{FAskR}_0\right] \\
&= \sum_{i=1}^{q_{\text{GG}}} \Pr[\text{GAskR}_0^i | \neg \text{FAskR}_0] \\
&= \sum_{i=1}^{q_{\text{GG}}} \Pr\left[\text{GAskR}_0^i \wedge \bigwedge_{j=1}^{i-1} \neg \text{GAskR}_0^j \mid \neg \text{FAskR}_0\right] \\
&= \sum_{i=1}^{q_{\text{GG}}} \Pr\left[\bigwedge_{j=1}^{i-1} \neg \text{GAskR}_0^j \mid \neg \text{FAskR}_0\right] \cdot \Pr\left[\text{GAskR}_0^i \mid \bigwedge_{j=1}^{i-1} \neg \text{GAskR}_0^j \wedge \neg \text{FAskR}_0\right] \\
&\leq \sum_{i=1}^{q_{\text{GG}}} \Pr\left[\text{GAskR}_0^i \mid \bigwedge_{j=1}^{i-1} \neg \text{GAskR}_0^j \wedge \neg \text{FAskR}_0\right].
\end{aligned}$$

**Lemma 1.** *For any  $i$  between 1 and  $q_{\text{GG}}$ ,*

$$\Pr\left[\text{GAskR}_0^i \mid \bigwedge_{j=1}^{i-1} \neg \text{GAskR}_0^j \wedge \neg \text{FAskR}_0\right] \leq 2^{-l+1} E_{H'}[M_{H'}].$$

Here, and everywhere below,  $E_{H'}[M_{H'}]$  means  $E[H' \xleftarrow{R} \Omega : M_{H'}]$ , i.e., the expected value of  $M_{H'}$  for  $H'$  distributed as a random function with input size  $l_s$  and output size  $k_0 - l_t$ .

*Proof.* Let  $r_A$  denote the coins of  $A$ . Let  $r_1^f, \dots, r_{q_{\text{FG}}}^f, r_1^g, \dots, r_{i-1}^g, r_i^g$  denote  $A$ 's queries to  $G$  in the find and guess stages, in order, up to and including the  $i$ th query in the guess stage. Let  $G_1^f, \dots, G_{q_{\text{FG}}}^f, G_1^g, \dots, G_{i-1}^g$  denote the responses to those queries, in order, except the last one.

Let's fix some values of  $H, r_A, r_1^f, \dots, r_{q_{\text{FG}}}^f, G_1^f, \dots, G_{q_{\text{FG}}}^f, x_0, x_1, c_f, b, r_1^g, \dots, r_i^g, G_1^g, \dots$ , and  $G_{i-1}^g$  that do not contradict  $\text{EqCond}$  (i.e., the probability of those values given  $\text{EqCond}$  is non-zero). Here  $\text{EqCond}$  is the event that the values that we have fixed for the variables are consistent among themselves and with  $\tilde{y}$ . More precisely,

we define

EqCond =

$$\begin{aligned} & \left( (r_1^f = \dots) \wedge \dots \wedge (r_j^f = A^H(r_A, L, \text{find}, j, G_1^f, \dots, G_{j-1}^f)) \wedge \dots \wedge (r_{q_{\text{FG}}}^f = \dots) \right. \\ & \quad \wedge ((x_0, x_1, c_f) = A^H(r_A, L, \text{find}, G_1^f, \dots, G_{q_{\text{FG}}}^f)) \\ & \quad \left. \wedge (r_1^g = \dots) \wedge \dots \wedge (r_j^g = A^H(r_A, \tilde{y}, c_f, \text{guess}, j, G_1^g, \dots, G_{j-1}^g)) \wedge \dots \wedge (r_i^g = \dots) \right). \end{aligned}$$

Here we have written out explicitly all of  $A$ 's inputs, including the random coins and the responses to the queries to  $G$ , omitting only the responses to the queries to  $H$ . The notation  $A^H(\dots, \text{stage}, j, \dots)$ , where  $\text{stage} \in \{\text{find}, \text{guess}\}$  and  $j$  is an integer, means  $A$ 's  $j$ th query to  $G$  in the corresponding stage.

We wish to bound

$$\Pr[G \stackrel{R}{\leftarrow} \Omega; r_0 \stackrel{R}{\leftarrow} \{0, 1\}^{k_0} : \text{GAskR}_0^i \mid \bigwedge_{j=1}^{i-1} \neg \text{GAskR}_0^j \wedge \neg \text{FAskR}_0 \wedge \text{EqCond}].$$

Let  $\text{NR}_0$  denote the condition  $r_0 \notin \{r_1^f, \dots, r_{q_{\text{FG}}}^f, r_1^g, \dots, r_{i-1}^g\}$ . Let  $\text{EqNR}_0$  denote  $\text{EqCond} \wedge \text{NR}_0$ . Simplifying and splitting over the possible values of  $\tilde{y}$ , we get

$$\begin{aligned} & \Pr[G \stackrel{R}{\leftarrow} \Omega; r_0 \stackrel{R}{\leftarrow} \{0, 1\}^{k_0} : \text{GAskR}_0^i \mid \bigwedge_{j=1}^{i-1} \neg \text{GAskR}_0^j \wedge \neg \text{FAskR}_0 \wedge \text{EqCond}] \\ &= \Pr[r_0 \stackrel{R}{\leftarrow} \{0, 1\}^{k_0}; G(r_0) \stackrel{R}{\leftarrow} \{0, 1\}^n; \tilde{y} \leftarrow h_{n', L}(\text{OAEP}^{G, H}(x_b, r_0)) : r_i^g = r_0 \mid \text{EqNR}_0] \\ &= \sum_{\tilde{y} \in \{0, 1\}^{n'-l}} \Pr[r_0 \stackrel{R}{\leftarrow} \{0, 1\}^{k_0}; G(r_0) \stackrel{R}{\leftarrow} \{0, 1\}^n : \tilde{y} = h_{n', L}(\text{OAEP}^{G, H}(x_b, r_0)) \wedge \\ & \quad r_i^g = r_0 \mid \text{EqNR}_0]. \end{aligned}$$

We now turn the probability of the conjunction  $\tilde{y} = h_{n', L}(\text{OAEP}^{G, H}(x_b, r_0)) \wedge r_i^g =$

$r_0$  into a product of two probabilities:

$$\begin{aligned}
& \sum_{\tilde{y} \in \{0,1\}^{n'-l}} \Pr[r_0 \stackrel{R}{\leftarrow} \{0,1\}^{k_0}; G(r_0) \stackrel{R}{\leftarrow} \{0,1\}^n : \tilde{y} = h_{n',L}(\text{OAEP}^{G,H}(x_b, r_0)) \wedge r_i^{\mathbf{g}} = r_0 | \\
& \quad \text{EqNR}_0] \\
&= \sum_{\tilde{y} \in \{0,1\}^{n'-l}} \Pr[r_0 \stackrel{R}{\leftarrow} \{0,1\}^{k_0}; G(r_0) \stackrel{R}{\leftarrow} \{0,1\}^n : r_i^{\mathbf{g}} = r_0 | \text{EqNR}_0] \times \\
& \quad \Pr[r_0 \stackrel{R}{\leftarrow} \{0,1\}^{k_0}; G(r_0) \stackrel{R}{\leftarrow} \{0,1\}^n : \tilde{y} = h_{n',L}(\text{OAEP}^{G,H}(x_b, r_0)) | \\
& \quad r_i^{\mathbf{g}} = r_0 \wedge \text{EqNR}_0]
\end{aligned}$$

To evaluate  $\Pr[r_i^{\mathbf{g}} = r_0 | \text{EqNR}_0]$  for a fixed  $\tilde{y}$ , we note that  $r_0$  is uniformly distributed over the set  $\{0,1\}^{k_0} \setminus \{r_1^{\mathbf{f}}, \dots, r_{q_{\text{FG}}}^{\mathbf{f}}, r_1^{\mathbf{g}}, \dots, r_{i-1}^{\mathbf{g}}\}$  (which is, at the moment, fixed), and  $r_i^{\mathbf{g}}$  is an element of that set (since queries are unique). Also,  $r_0$  is independent of  $r_i^{\mathbf{g}}$  (since, at the moment,  $\tilde{y}$  is fixed). Therefore,

$$\Pr[r_0 \stackrel{R}{\leftarrow} \{0,1\}^{k_0}; G(r_0) \stackrel{R}{\leftarrow} \{0,1\}^n : r_i^{\mathbf{g}} = r_0 | \text{EqNR}_0] = (2^{k_0} - q_{\text{FG}} - i + 1)^{-1}.$$

Now let's bound  $\Pr[\tilde{y} = h_{n',L}(\text{OAEP}^{G,H}(x_b, r_0)) | r_i^{\mathbf{g}} = r_0 \wedge \text{EqNR}_0]$ :

$$\begin{aligned}
& \Pr[r_0 \stackrel{R}{\leftarrow} \{0,1\}^{k_0}; G(r_0) \stackrel{R}{\leftarrow} \{0,1\}^n : \tilde{y} = h_{n',L}(\text{OAEP}^{G,H}(x_b, r_0)) | r_i^{\mathbf{g}} = r_0 \wedge \text{EqNR}_0] \\
&= \Pr[r_0 \stackrel{R}{\leftarrow} \{0,1\}^{k_0}; G(r_0) \stackrel{R}{\leftarrow} \{0,1\}^n : r_0 \in \mathcal{R}^{G,H} | r_i^{\mathbf{g}} = r_0 \wedge \text{EqNR}_0] \\
&= \Pr[r_0 \stackrel{R}{\leftarrow} \{0,1\}^{k_0}; G(r_0) \stackrel{R}{\leftarrow} \{0,1\}^n : x_b \oplus G(r_0) \in \mathcal{S}_{r_0}^H | r_i^{\mathbf{g}} = r_0 \wedge \text{EqNR}_0] \\
&= \Pr[G(r_i^{\mathbf{g}}) \stackrel{R}{\leftarrow} \{0,1\}^n : G(r_i^{\mathbf{g}}) \in x_b \oplus \mathcal{S}_{r_i^{\mathbf{g}}}^H | \text{EqCond}] \\
&= 2^{-n} |x_b \oplus \mathcal{S}_{r_i^{\mathbf{g}}}^H| \\
&= 2^{-n} |\mathcal{S}_{r_i^{\mathbf{g}}}^H| \\
&= 2^{-n} |H'^{\{-1\}}(h_t(r_i^{\mathbf{g}}) \oplus \tilde{t})| \\
&\leq 2^{-n} M_{H'}.
\end{aligned}$$

Here we have used the fact that  $G(r_i^{\mathbf{g}})$  is uniform and independent of  $x_b \oplus \mathcal{S}_{r_i^{\mathbf{g}}}^H$ , since

$r_i^g$  is distinct from all of  $A$ 's other queries to  $G$ , and since  $\tilde{y}$  is, at the moment, fixed.

We now get

$$\begin{aligned}
& \Pr[G \stackrel{R}{\leftarrow} \Omega; r_0 \stackrel{R}{\leftarrow} \{0, 1\}^{k_0} : \text{GAskR}_0^i \mid \bigwedge_{j=1}^{i-1} \neg \text{GAskR}_0^j \wedge \neg \text{FAskR}_0 \wedge \text{EqCond}] \\
&= \sum_{\tilde{y} \in \{0, 1\}^{n'-l}} \Pr[r_0 \stackrel{R}{\leftarrow} \{0, 1\}^{k_0}; G(r_0) \stackrel{R}{\leftarrow} \{0, 1\}^n : r_i^g = r_0 \mid \text{EqNR}_0] \times \\
&\quad \Pr[r_0 \stackrel{R}{\leftarrow} \{0, 1\}^{k_0}; G(r_0) \stackrel{R}{\leftarrow} \{0, 1\}^n : \tilde{y} = h_{n', L}(\text{OAEP}^{G, H}(x_b, r_0)) \mid \\
&\quad \quad \quad r_i^g = r_0 \wedge \text{EqNR}_0] \\
&\leq \sum_{\tilde{y} \in \{0, 1\}^{n'-l}} (2^{k_0} - q_{\text{FG}} - i + 1)^{-1} \times 2^{-n} M_{H'} \\
&= 2^{-n} (2^{k_0} - q_{\text{FG}} - i + 1)^{-1} \sum_{\tilde{y} \in \{0, 1\}^{n'-l}} M_{H'}.
\end{aligned}$$

Let  $\Sigma = \sum_{\tilde{y} \in \{0, 1\}^{n'-l}} M_{H'}$ . We have

$$\begin{aligned}
2^{-n} (2^{k_0} - q_{\text{FG}} - i + 1)^{-1} \Sigma &\leq 2^{-n} (2^{k_0} - q_{\text{FG}} - q_{\text{GG}})^{-1} \Sigma \\
&= 2^{-n} (2^{k_0} - q_{\text{G}})^{-1} \Sigma \\
&\leq 2^{-n} (2^{k_0} - 2^{k_0-1})^{-1} \Sigma \\
&= 2^{-n-k_0+1} \Sigma
\end{aligned}$$

(since  $q_{\text{G}} \leq 2^{k_0-1}$ ).

Taking the probability over  $H$  and the other random variables that we have fixed,

we get

$$\begin{aligned}
\Pr[\text{GAskR}_0^i | \bigwedge_{j=1}^{i-1} \neg \text{GAskR}_0^j \wedge \neg \text{FAskR}_0] &\leq 2^{-n-k_0+1} E[\Sigma] \\
&= 2^{-n-k_0+1} \sum_{\tilde{y} \in \{0,1\}^{n'-l}} E_H[M_{H'}] \\
&= 2^{-n-k_0+1} \sum_{\tilde{y} \in \{0,1\}^{n'-l}} E_{H'}[M_{H'}] \\
&= 2^{-n-k_0+1} \cdot 2^{n+k_0-l} E_{H'}[M_{H'}] \\
&= 2^{-l+1} E_{H'}[M_{H'}].
\end{aligned}$$

Here  $E_H[M_{H'}] = E_{H'}[M_{H'}]$  for any fixed  $\tilde{y}$ , since, as noted above,  $H'$  is a random function whenever  $H$  is a random function and  $\tilde{y}$  is fixed.  $\square$

We now have

$$\Pr[\text{GAskR}_0 | \neg \text{FAskR}_0] \leq q_{\text{GG}} 2^{-l+1} E_{H'}[M_{H'}]$$

and

$$\begin{aligned}
\Pr[\text{AC}] &\leq \Pr[\text{AC} | \neg \text{AskR}_0] + \Pr[\text{FAskR}_0] + \Pr[\text{GAskR}_0 | \neg \text{FAskR}_0] \\
&\leq \frac{1}{2} + q_{\text{FG}} 2^{-k_0} + q_{\text{GG}} 2^{-l+1} E_{H'}[M_{H'}].
\end{aligned}$$

Let's compute  $E_{H'}[M_{H'}]$ . We will need to show

**Lemma 2.** *Let  $F : U \rightarrow U$  be a random function, for a set  $U$  of size  $N > e$ . Then*

$$E_F[\max_{y \in U} |F^{\{-1\}}(y)|] \leq \frac{e \ln N}{\ln \ln N} + 1.$$

*Proof.* The following result appears in Motwani and Raghavan [73, Theorem 3.1]:

**Lemma 3.** *Suppose  $N$  balls are randomly assigned to  $N$  bins. Then, with probability at least  $1 - 1/N$ , no bin will have more than  $k^* = \frac{e \ln N}{\ln \ln N}$  balls in it.*

It is easy to see that this implies the following:

**Lemma 4.** *Let  $F : U \rightarrow U$  be a random function, for a set  $U$  of size  $N$ . Let  $k^* = \frac{e \ln N}{\ln \ln N}$ . Then*

$$\Pr[\forall y \in U : |F^{\{-1\}}(y)| \leq k^*] \geq 1 - 1/N. \quad (2.3)$$

The two statements are equivalent, since the random function  $F$  corresponds to the random assignment of “balls” (inputs) to “bins” (outputs).

Let  $M_F = \max_{y \in U} |F^{\{-1\}}(y)|$ . We can restate (2.3) as follows:

$$\Pr[M_F > k^*] \leq 1/N.$$

Clearly,  $M_F \leq N$ . We have

$$E_F[M_F] \leq k^* \Pr[M_F \leq k^*] + N \Pr[M_F > k^*] \leq k^* + N \cdot \frac{1}{N} = k^* + 1.$$

Thus  $E_F[M_F] \leq \frac{e \ln N}{\ln \ln N} + 1$ , as we wished to show.  $\square$

**Corollary 1.** *Let  $F : \{0, 1\}^m \rightarrow \{0, 1\}^n$  be a random function, for  $m \leq n$  and  $n > 1$ . Then  $E_F[\max_{y \in \{0, 1\}^n} |F^{\{-1\}}(y)|] \leq \frac{e \ln 2^n}{\ln \ln 2^n} + 1$ .*

This corollary easily follows, since the maximum number of balls in a bin will not increase if we decrease the total number of balls.

We have  $l_s \leq k_0 - l_t$  (we assume that  $l \leq k_0$ ). Suppose  $k_0 - l_t > 1$ . Then

$$\begin{aligned} \Pr[\text{AC}] &\leq \frac{1}{2} + q_{\text{FG}} 2^{-k_0} + q_{\text{GG}} 2^{-l+1} E_{H'}[M_{H'}] \\ &\leq \frac{1}{2} + q_{\text{FG}} 2^{-k_0} + q_{\text{GG}} 2^{-l+1} \left( \frac{e \ln 2^{k_0-l_t}}{\ln \ln 2^{k_0-l_t}} + 1 \right). \end{aligned}$$

We will use the fact that  $x \mapsto \frac{\ln 2^x}{\ln \ln 2^x}$  is a monotonically increasing function for  $x \geq \log_2(e^e) = 3.92 \dots$ . Taking into account the possibility that  $k_0 - l_t$  may be 2 or 3, after a simple calculation we get

$$\frac{e \ln 2^{k_0-l_t}}{\ln \ln 2^{k_0-l_t}} \leq \frac{e \ln 2^{k_0}}{\ln \ln 2^{k_0}},$$

assuming  $k_0 \geq 14$ . Now

$$\frac{e \ln 2^{k_0}}{\ln \ln 2^{k_0}} + 1 = \frac{k_0 e \ln 2}{\ln k_0 + \ln \ln 2} + 1 = \frac{k_0 e}{\log_2 k_0 + \log_2 \ln 2} + 1 \leq \frac{4k_0}{\log_2 k_0}$$

for all  $k_0 \geq 5$ . Thus

$$\begin{aligned} \Pr[\text{AC}] &\leq \frac{1}{2} + q_{\text{FG}} 2^{-k_0} + q_{\text{GG}} 2^{-l+1} \frac{4k_0}{\log_2 k_0} \\ &\leq \frac{1}{2} + (q_{\text{FG}} + q_{\text{GG}}) \max(2^{-k_0}, 2^{-l+3} \frac{k_0}{\log_2 k_0}) \\ &= \frac{1}{2} + q_{\text{G}} \max(2^{-k_0}, 2^{-l+3} \frac{k_0}{\log_2 k_0}). \end{aligned}$$

Since  $l \leq k_0$ , we have  $2^{-k_0} \leq 2^{-l}$ , and so

$$\Pr[\text{AC}] \leq \frac{1}{2} + 8q_{\text{G}} \frac{k_0}{\log_2 k_0} 2^{-l}.$$

On the other hand, if  $k_0 - l_t \in \{0, 1\}$ , then  $M_{H'} \leq 2$ . Therefore,

$$\begin{aligned} \Pr[\text{AC}] &\leq \frac{1}{2} + q_{\text{FG}} 2^{-k_0} + 2^{-l+1} q_{\text{GG}} E_H[M_{H'}] \\ &\leq \frac{1}{2} + q_{\text{FG}} 2^{-k_0} + 2^{-l+2} q_{\text{GG}} \\ &\leq \frac{1}{2} + (q_{\text{FG}} + q_{\text{GG}}) \max(2^{-k_0}, 2^{-l+2}) \\ &= \frac{1}{2} + q_{\text{G}} \max(2^{-k_0}, 2^{-l+2}) \\ &= \frac{1}{2} + 4q_{\text{G}} 2^{-l} \\ &\leq \frac{1}{2} + 8q_{\text{G}} \frac{k_0}{\log_2 k_0} 2^{-l} \end{aligned}$$

(since  $2 \frac{k_0}{\log_2 k_0} \geq 1$  for all  $k_0 \geq 2$ ).

Thus we have in both cases

$$\Pr[\text{AC}] \leq \frac{1}{2} + 8q_{\text{G}} \frac{k_0}{\log_2 k_0} 2^{-l},$$

which is what we wished to show.

## 2.5.2 Proof Outline for Theorem 2

The exhaustive search algorithm is as follows:

1. **Find stage:** Choose  $x_0$  and  $x_1$  independently at random from  $\{0, 1\}^n$ . Let  $c_f = x_0 \parallel x_1$ . Output  $x_0$ ,  $x_1$ , and  $c_f$ .
2. **Guess stage:** Let  $\tilde{y}$  be the value of  $h_{n', L}(y)$  that is given as input. Extract  $x_0$  and  $x_1$  from  $c_f$ . Repeat  $N$  times:
  - Choose random  $v \in \{0, 1\}^l$ . Compute  $x = (\text{AONT}^\Gamma)^{-1}(u_{n', L}(v, \tilde{y}))$ . If  $x = x_{b'}$  for some  $b' \in \{0, 1\}$ , then output  $b'$  and halt (we will call this “success,” whether or not  $b' = b$ ). Otherwise, continue.

If there has been no success after  $N$  trials, then output a random  $b' \in \{0, 1\}$ .

This algorithm evaluates AONT at most  $N$  times, so it takes at most  $NT$  time and makes at most  $Nq_\Gamma$  queries to  $\Gamma$ .

Let's estimate the advantage given by this algorithm over blindly guessing  $b$ . Let **AC** be the event that the algorithm outputs the correct answer. Let **SuccC** (**SuccI**) be the event that one of the trials succeeds and gives the correct (incorrect) output.

It is easy to see that

$$\Pr[\text{AC}] = \frac{1}{2}(1 + \Pr[\text{SuccC}] - \Pr[\text{SuccI}]).$$

Let **Unique** be the event that  $x_{1-b}$  is not consistent with  $\tilde{y}$  (i.e., there is no  $v$  such that  $x_{1-b} = (\text{AONT}^\Gamma)^{-1}(u_{n', L}(v, \tilde{y}))$ ). In other words, **Unique** means that only one of  $x_0$  and  $x_1$  is consistent with  $\tilde{y}$ , since  $x_b$  is consistent with it by construction. If **Unique** is true, then no trial can succeed with  $1 - b$ . Also, each trial has probability at least  $2^{-l}$  of succeeding with  $b$  (since there is at least one good  $v$ ). Using inclusion-exclusion, this easily leads to the bound  $\Pr[\text{SuccC} | \text{Unique}] \geq 2^{-l-1}N$  (since there are  $N < 2^l$  independent trials, each with success probability of at least  $2^{-l}$ ).

We have  $\Pr[\text{Unique}] \geq 1 - 2^{l-n} \geq \frac{1}{2}$ , since there are at most  $2^l$  values of  $x$  corresponding to the  $2^l$  possible values of  $y$  that are consistent with  $\tilde{y}$ , and they are

independent of  $x_{1-b}$ , which is chosen at random out of a set of size  $2^n$ . Therefore,

$$\Pr[\text{SuccC}] \geq \Pr[\text{Unique}] \cdot \Pr[\text{SuccC}|\text{Unique}] \geq \Pr[\text{Unique}] \cdot 2^{-l-1}N \geq 2^{-l-2}N.$$

Let's bound  $\Pr[\text{SuccI}]$ . For any particular trial,

$$\Pr[x_{1-b} = (\text{AONT}^\Gamma)^{-1}(u_{n',L}(v, \tilde{y}))] = 2^{-n},$$

since  $x_{1-b}$  is uniform and independent of all the other variables. Therefore,  $\Pr[\text{SuccI}] \leq 2^{-n}N$ .

We now get

$$\Pr[\text{AC}] = \frac{1}{2}(1 + \Pr[\text{SuccC}] - \Pr[\text{SuccI}]) \geq \frac{1}{2}(1 + 2^{-l-2}N - 2^{-n}N) \geq \frac{1}{2} + \frac{1}{16}N2^{-l},$$

since  $l \leq n - 3$ . This is what we wished to show.

### 2.5.3 Proof Outline for Theorem 3

The proof is almost the same as for the non-adaptive indistinguishability scenario. The only major change is that it is no longer obvious that  $H'$  can be viewed as a random function. The reason is that the adversary may now choose  $L$  using some information about  $H$  ( $A$  can query  $H$  in the select stage). Therefore, if we fix a value of  $L$ ,  $H$  and its restriction  $H'$  might no longer be distributed as random functions. The idea of the solution to this problem is that, with overwhelming probability, none of the queries to  $H$  made in the select stage will be in  $\mathcal{S}$ , which is the only part of the domain of  $H$  that affects  $H'$ . Then we will still be able to view  $H'$  as a random function.

Let  $q_{\text{SG}}$  and  $q_{\text{SH}}$  be the number of  $A$ 's queries to  $G$  and  $H$ , respectively, in the select stage (without loss of generality, we can assume that these are fixed for any given  $n$  and  $k_0$ ). Let  $\text{SAskR}_0$  be the event that query  $r_0$  has been asked of  $G$  in the select stage. Let  $\text{SAskS}$  be the event that a query made by  $A$  to  $H$  in the select stage

is an element of  $\mathcal{S}$ . Let  $\text{SFAAskR}_0 = \text{SAskR}_0 \vee \text{FAAskR}_0$ . We have

$$\begin{aligned} \Pr[\text{AC}] &\leq \Pr[\text{AC}|\neg\text{AskR}_0] + \Pr[\text{SFAAskR}_0] + \Pr[\text{GAskR}_0|\neg\text{SFAAskR}_0] \\ &\leq \Pr[\text{AC}|\neg\text{AskR}_0] + \Pr[\text{SFAAskR}_0] + \Pr[\text{SAskS}|\neg\text{SFAAskR}_0] \\ &\quad + \Pr[\text{GAskR}_0|\neg\text{SFAAskR}_0 \wedge \neg\text{SAskS}]. \end{aligned}$$

Similarly to the non-adaptive case, it is easy to see that

$$\begin{aligned} \Pr[\text{AC}|\neg\text{AskR}_0] &= \frac{1}{2}, \\ \Pr[\text{SFAAskR}_0] &\leq (q_{\text{SG}} + q_{\text{FG}})2^{-k_0}. \end{aligned}$$

Let's bound  $\Pr[\text{SAskS}|\neg\text{SFAAskR}_0]$ . Suppose  $\neg\text{SFAAskR}_0$ . Then  $\tilde{s} = h_s(G(r_0)) \oplus h_s(x_b)$  will be distributed uniformly over  $2^{n-l_s}$ , and independently from any of  $A$ 's inputs in the select stage. The probability that any particular query  $s'$  made to  $H$  in the select stage is in  $\mathcal{S}$ , i.e., the probability that  $h_s(s') = \tilde{s}$ , is then  $2^{l_s-n}$ . Consequently,

$$\Pr[\text{SAskS}|\neg\text{SFAAskR}_0] \leq q_{\text{SH}}2^{l_s-n}.$$

Now let's estimate  $\Pr[\text{GAskR}_0|\neg\text{SFAAskR}_0 \wedge \neg\text{SAskS}]$ . As before, let  $r_A$  denote the coins of  $A$ . Let  $s_1^s, \dots, s_{q_{\text{SH}}}^s$  denote  $A$ 's queries to  $H$  in the select stage. Let  $H_1^s, \dots, H_{q_{\text{SH}}}^s$  denote the responses to those queries.

Let's fix some values of  $s_1^s, \dots, s_{q_{\text{SH}}}^s, H_1^s, \dots, H_{q_{\text{SH}}}^s$ , and  $L$ , that satisfy  $\neg\text{SAskS}$ . As before, let  $\text{GAskR}_0^i$  denote the event that  $A$ 's  $i$ th query to  $G$  in the guess stage is  $r_0$ . Similarly to Lemma 1, we can show the following:

**Lemma 5.** *For any  $i$  between 1 and  $q_{\text{GG}}$ ,*

$$\Pr[\text{GAskR}_0^i | \bigwedge_{j=1}^{i-1} \neg\text{GAskR}_0^j \wedge \neg\text{SFAAskR}_0 \wedge \neg\text{SAskS}] \leq 2^{-l+1} E_{H'}[M_{H'}].$$

*outline.* We can use the proof of Lemma 1, with trivial modifications, except that now we need a new proof for the equality  $E_H[M_{H'}] = E_{H'}[M_{H'}]$ , for any fixed  $\tilde{y}$ .

The distribution of  $H$  is as follows: It is constrained to values  $H_1^s, \dots, H_{q_{\text{SH}}}^s$  on inputs  $s_1^s, \dots, s_{q_{\text{SH}}}^s$ . However, on the domain  $\{0, 1\}^n \setminus \{s_j^s\}_{j=1}^{q_{\text{SH}}}$ ,  $H$  is distributed as a random function. Let  $H_{\mathcal{S}}$  be the restriction of  $H$  to domain  $\mathcal{S}$ . Since  $\neg\text{SAskS}$ , we have  $\mathcal{S} \subseteq \{0, 1\}^n \setminus \{s_j^s\}_{j=1}^{q_{\text{SH}}}$ . Therefore,  $H_{\mathcal{S}}$  is distributed as a random function. Since  $H'$  can be written as

$$H'(v) = h_t(H_{\mathcal{S}}(u_s(v, \tilde{s})))$$

(because  $u_s(v, \tilde{s}) \in \mathcal{S}$  for any  $v$ ),  $H'$  is also distributed as a random function, for any fixed  $\tilde{y}$ . Consequently,  $E_H[M_{H'}] = E_{H'}[M_{H'}]$  for any fixed  $\tilde{y}$ .  $\square$

We now have

$$\Pr[\text{GAskR}_0 | \neg\text{SAskR}_0 \wedge \neg\text{SAskS}] \leq q_{\text{GG}} 2^{-l+1} E_{H'}[M_{H'}].$$

and

$$\begin{aligned} \Pr[\text{AC}] &\leq \Pr[\text{AC} | \neg\text{AskR}_0] + \Pr[\text{SAskR}_0] + \Pr[\text{SAskS} | \neg\text{SAskR}_0] + \\ &\quad \Pr[\text{GAskR}_0 | \neg\text{SAskR}_0 \wedge \neg\text{SAskS}] \\ &\leq \frac{1}{2} + (q_{\text{SG}} + q_{\text{FG}}) 2^{-k_0} + q_{\text{SH}} 2^{l_s - n} + q_{\text{GG}} 2^{-l+1} E_{H'}[M_{H'}] \\ &\leq \frac{1}{2} + q_{\text{H}} 2^{l-n} + q_{\text{G}} \max(2^{-k_0}, 2^{-l+1} E_{H'}[M_{H'}]) \\ &\leq \frac{1}{2} + (q_{\text{G}} + q_{\text{H}}) \max(2^{l-n}, 2^{-k_0}, 2^{-l+1} E_{H'}[M_{H'}]). \end{aligned}$$

Since  $l \leq \frac{n}{2}$ , we have  $2^{l-n} \leq 2^{-l}$ . Also, since  $l \leq k_0$ , we have  $2^{-k_0} \leq 2^{-l}$ . Therefore,

$$\Pr[\text{AC}] \leq \frac{1}{2} + (q_{\text{G}} + q_{\text{H}}) 2^{-l+1} \max(1, E_{H'}[M_{H'}]).$$

Using Corollary 1, similarly to the non-adaptive case, we get

$$\Pr[\text{AC}] \leq \frac{1}{2} + 8(q_{\text{G}} + q_{\text{H}}) \frac{k_0}{\log_2 k_0} 2^{-l},$$

which is what we wished to show.

### 2.5.4 Proof Outline for Theorem 4

We can use the proof of Theorem 1 with the following simple modifications: We need to substitute  $x$  for  $x_b$ . Those queries in the guess stage that have been asked in the find stage need to be answered in the same way as they were the first time. Also,  $\text{GAskR}_0^i$  will now refer to  $A$ 's  $i$ th query to  $G$  in the guess stage, *of all the queries to  $G$  that have not been asked in the find stage.*

The only change that affects the bound on  $\Pr[\text{AC}]$  is that the estimate on  $\Pr[\text{AC}|\neg\text{AskR}_0]$  is different from Theorem 1.

**Lemma 6.** *Let  $X$  be an arbitrary distribution. If a random variable  $f'$  is independent of  $x$ , then*

$$\Pr[x \stackrel{R}{\leftarrow} X : f' = f(x)] \leq \max_y (\Pr[x \stackrel{R}{\leftarrow} X : f(x) = y]).$$

*Proof.*

$$\begin{aligned} \Pr[x \stackrel{R}{\leftarrow} X : f' = f(x)] &= \sum_y \Pr[x \stackrel{R}{\leftarrow} X : f' = y \wedge f(x) = y] \\ &= \sum_y \Pr[f' = y] \Pr[x \stackrel{R}{\leftarrow} X : f(x) = y] \\ &\leq \sum_y \Pr[f' = y] \max_y \Pr[x \stackrel{R}{\leftarrow} X : f(x) = y] \\ &= \max_y (\Pr[x \stackrel{R}{\leftarrow} X : f(x) = y]) \sum_y \Pr[f' = y] \\ &= \max_y (\Pr[x \stackrel{R}{\leftarrow} X : f(x) = y]). \end{aligned}$$

□

It follows that  $\Pr[\text{AC}|\neg\text{AskR}_0] \leq p_{A,f}$ , since if  $A$  does not see  $G(r_0)$ , all of its inputs will be independent of  $x$ .

Continuing as in the proof of Theorem 1, we get

$$\Pr[\text{AC}] \leq p_{A,f} + 8q_G \frac{k_0}{\log_2 k_0} 2^{-l},$$

which is what we wished to show.

## 2.6 Conclusions and Open Problems

We have presented new and very strong formal definitions of security for AONTs. We have proposed OAEP as a candidate AONT, and shown that it satisfies these definitions. The bounds that we have proved are nearly optimal, in the sense that no adversary can do much better against OAEP than exhaustive search, and in the sense that no AONT construction can achieve a substantially better bound than the one we have shown for OAEP.

We note that an AONT that satisfies our definitions, such as OAEP, can be used to implement the “puzzles” of Merkle [70], or the notion of uniform security of Even et al. [42]. The “puzzles” could be made by publishing a certain number of bits of AONT output on a bit string of sufficient redundancy (so that, with overwhelming probability, only one such string corresponded to the published information). Similarly, cryptograms of uniform security could be achieved by publishing a part of AONT output on the key used in the cryptogram. It seems, though, that a simpler construction would suffice: Let  $E(K, M) \rightarrow C$  be a regular symmetric cryptosystem, and let  $H$  be a random oracle. Then, it seems to us that, using the methods of this thesis, it can be shown that  $E(H(K), M) \rightarrow C$  is a uniformly secure cryptosystem. On the other hand, the AONT construction for the “puzzles” has the advantage that it does not use encryption, which could put it outside the scope of export regulations. It would be interesting to investigate these issues further.

The first open problem that comes to mind is to improve our bounds. The best would be to bring the upper bounds within a constant factor of exhaustive search, or to devise an algorithm that does better than exhaustive search. Also, it would be interesting to give lower bounds for the semantic security scenarios.

Another open problem is to show equivalence (or non-equivalence) of our definitions, trying to carry over the exact bounds as much as possible. There are also other possible models to consider, such as the fully-adaptive scenario described in

Section 2.3.1. Also, just as there are several variations of the definition of semantic security for encryption, one might consider other definitions for AONTs, and whether they are equivalent to the ones in this thesis. We note that the subsequent paper Canetti et al. [23] states that all their constructions satisfy fully-adaptive security, and it is only due to the messiness of an appropriate definition that no formal definition is given. Unfortunately, we don't see any trivial way to extend our proofs to such a scenario, so the security of OAEP in that context is thus an open problem. This is just one of the open questions related to the use of OAEP in AONT applications. Another important open question related to applications is the security of AONT-based encryption against adaptive chosen-ciphertext attacks.

One of the most interesting problems related to AONTs is to construct a secure AONT without the use of random oracles. As mentioned above in Section 2.1.4, this problem has been partially solved by Canetti et al. [23]. However, as also mentioned above, their constructions do not appear to be useful for the original application of AONTs, i.e., protection against exhaustive key search. That application has been addressed by Desai [34]. However, his construction, while not relying on random oracles, does use the non-standard ideal block cipher assumption. Thus, there is still an open problem concerning the construction of a secure AONT based on standard computational assumptions, if we want it to be useful for the original application. Canetti et al. [34], besides the constructions mentioned above, also give a result that may be of help in solving this problem: Consider a variant of OAEP, where the function  $H$  is replaced by the identity function, and  $G$  is any function such that both  $G(r)$  and  $G(r) \oplus r$  are length-preserving ERFs (see Section 2.1.4). Then the resulting transform is shown to be an AONT. This result does not offer an immediate solution, for two reasons: First, no deterministic choice of  $G$  is known that would satisfy the above property. Second, the actual parameters of the construction result in overhead equal to the length of the message, which is unacceptable (as mentioned in Section 2.1.4, if we are willing to tolerate such overhead, then we can settle for a much simpler construction that is also based on standard assumptions). In any case, we hope that the above mentioned result will be a good base for further research.

Another interesting question is whether there is any relation between the properties of OAEP as an AONT, and its original proposed use for constructing secure cryptosystems. One might ask, for instance, if OAEP could be replaced by an arbitrary AONT in the construction of Bellare and Rogaway [10].

One could also look into the possibility of generalizing the definitions of AONT security, so that instead of getting a certain number of bits of the output, the adversary gets the equivalent amount of information through other means, i.e., by seeing the value of some transformation of the output that reduces its entropy by  $l$ . The function  $h_{n',L}$  is just one example of such a transformation.

Finally, now that we have a provably secure AONT, it would be of great interest to find new applications of this primitive. We hope that its usefulness extends far beyond its original applications.

As an example of a potential new application, it seems that an AONT might be used to construct a protocol for simultaneous exchange of information. The parties would apply an AONT to the messages that need to be exchanged. Then, in each round, they would send each other the next bit of their respective AONT outputs. If the protocol stops early, then the amounts of work needed by either party to reconstruct the other's message are different by at most a factor of  $2^{\frac{k_0}{\log_2 k_0}}$ , for the case of OAEP (since the number of bits of AONT output that the parties have differs by at most one, and a party can always apply exhaustive search). This computationally fair approach is along the lines of Even et al. [42].

In order to detect cheating in the above protocol, it would be necessary to use a zero-knowledge proof that the bits sent so far form a prefix of a valid AONT output on the message. It is unclear to us if such a zero-knowledge proof system can be made for an AONT that uses random oracles. However, it would certainly be possible for a computationally-secure AONT, if one is found with security that “smoothly” depends on the number of missing bits. We note that the model and constructions of Canetti et al. [23] do not consider security as a function of the number of missing bits, and are thus not directly suitable for this application.

# Chapter 3

## Provably Secure

## Password-Authenticated Key

## Exchange Using Diffie-Hellman

### 3.1 Introduction

Two entities, who only share a password, and who are communicating over an insecure network, want to authenticate each other and agree on a large session key to be used for protecting their subsequent communication. This is called the *password-authenticated key exchange* problem. If one of the entities is a user and the other is a server, then this can be seen as a problem in the area of *remote user access*. Many solutions for remote user access rely on cryptographically secure keys, and consequently have to deal with issues like key management, public-key infrastructure, or secure hardware. Many solutions that are password-based, like telnet or Kerberos, have problems that range from being totally insecure (telnet sends passwords in the clear) to being susceptible to certain types of attacks (Kerberos is vulnerable to off-line dictionary attacks [101]).

Over the past decade, many password-authenticated key exchange protocols that promised increased security have been developed, e.g., [13, 14, 50, 49, 95, 56, 57, 66,

100, 90].<sup>1</sup> Some of these have been broken [82, 83], and, in fact, only two very recent ones have been formally proven secure. The SNAP1 protocol in [67] is proven secure in the random oracle model,<sup>2</sup> assuming the security of RSA (and also Decision Diffie-Hellman,<sup>3</sup> when perfect forward secrecy is desired). The simple and elegant protocol in [7] is proven as secure as Decision Diffie-Hellman in a model that includes random oracles and ideal block ciphers. (Our work was performed independently of [7]. In fact, the conference version of our results [21] appears in the same proceedings as [7].)

We present a new password-authenticated key exchange protocol called **PAK** (Password-Authenticated Key exchange), which we prove to be as secure as Decision Diffie-Hellman in the random oracle model. Compared to the protocol of [67], PAK (1) does not require the RSA assumption for security, (2) is more efficient in terms of the number of rounds, and (3) is conceptually simpler, with a simpler proof. Compared to the protocol of [7], PAK does not require an ideal block cipher assumption for security, but has a more complicated proof. (We note that the ideal block cipher assumption is used much less often in the literature than the random oracle assumption.) We also show how the security of PAK can be related to the Computational Diffie-Hellman problem, although with weaker security bounds.

In addition to PAK, we also show a more efficient protocol called PPK (Password Protected Key exchange) that is provably secure in the implicit-authentication model. The PPK protocol only requires 2 rounds of communication.

We then extend PAK to a protocol called PAK-X, in which one side (the client) stores a plaintext version of the password, while the other side (the server) only stores a verifier for the password. We formally prove security of PAK-X, even when the server is compromised. Security in this case refers to an attacker not being able

---

<sup>1</sup>We will discuss hybrid protocols (i.e., password-based protocols in which a server public key is also known to the user) in Section 3.2.2.

<sup>2</sup>The random oracle model was introduced in [9]. Many popular protocols have been proven secure in that model, including Optimal Asymmetric Encryption Padding (OAEP) [10]. It would certainly be desirable to have a security proof using only standard cryptographic assumptions [24], but, so far, no protocol (and in particular, no efficient protocol) is known for the password authentication problem that is provably secure in the standard model.

<sup>3</sup>The hardness of the Decision Diffie-Hellman problem is essentially equivalent to the semantic security of the ElGamal encryption scheme [38]. See Boneh [18] for more information on this problem.

to pose as a client after compromising the server; naturally, it would be trivial to pose as the server.

Our formal model for password-authenticated key exchange is new, and may be of independent interest. It is based on the formal model for secure key exchange by Shoup [93] (which follows the work of [5]), enhanced with notions of password authentication security from [51, 67]. This model is based on the multi-party simulatability tradition (e.g. [3]), in which one first defines an ideal system that models, using a trusted center, the service to be performed (in this case, password-authenticated key exchange), and then one proves that the protocol running in the real world is essentially equivalent to that ideal system.

## 3.2 Background

### 3.2.1 User Authentication

Techniques for user authentication are broadly based on one or more of the following categories: (1) what a user knows, (2) what a user is, or (3) what a user has. Passwords or PINs are example of the first category. Biometric techniques, such as analysis of voice, fingerprints, retinal scans, or keystrokes, fit in the second category. Identification tokens, such as smart cards, fit in the third category. Techniques involving biometric devices tend to be cost-prohibitive, while techniques involving smart cards tend to be both expensive and relatively inconvenient for users. The least expensive and most convenient solutions for user authentication have been based on the first category, of “what a user knows,” and that is what we will focus on in this work.

In fact, we will focus on the harder problem of *remote user authentication*, in which not only must the basic authentication techniques be secure, but the protocol that communicates the authentication data across the network must also be secure. The need for remote user authentication is greatly increasing, due mainly to the explosive growth of the Internet and other types of networks, such as wireless communication networks. In any of these environments, it is safest to assume that the underlying

links or networks are insecure, and we should realistically expect that a powerful adversary would be capable of eavesdropping on legitimate sessions, deleting and inserting messages into those sessions, and even initiating sessions herself.

Now let us consider the question: “What can a user know?” It is common knowledge that users cannot remember long random numbers, hence if the user is required to know a large secret key (either a large symmetric key or a private key corresponding to a public key), then these keys will have to be stored on the user’s system. Furthermore, keeping these secret requires an extra security assumption and introduces a new point of weakness. Even if a user is required to know some public but non-generic data, like the server’s public key, this must be stored on the user’s system and requires an extra assumption that the public key cannot be modified. In either case, (1) there is a significant increase in administration overhead because both secret and public keys have to be generated and securely distributed to the user’s system and the server, and (2) this would not allow for users to walk up to a generic station that runs the authentication protocol and be able to perform secure remote authentication to a system that was previously unknown to that station (such as, perhaps, the user’s home system).

To solve these problems one may wish to use a trusted third party, either on-line (as in Kerberos) or off-line (i.e., a certification authority). However, the fact that the third party is “trusted” implies another security requirement. Also, the users or servers must at some point interact with the third party before they can communicate remotely, which increases the overhead of the whole system. Naturally, if an organized and comprehensive PKI emerges, this may be less of a problem. Still, password-only protocols seem very inviting because they are based on direct trust between a user and a server, and do not require the user to store long secrets or data on the user’s system. They are thus cheaper, more flexible, and less administration-intensive. They also allow for a generic protocol which can be pre-loaded onto users’ systems.

### 3.2.2 Password-Authentication Protocols

Many existing password authentication protocols, like `telnet` and `ftp`, send the password in the clear and are thus vulnerable to eavesdroppers.<sup>4</sup> Sending the password in the clear can be avoided by using more sophisticated schemes, such as one-time passwords systems (e.g., S/KEY [52]), or simple challenge-response schemes (e.g., CHAP [94]). However, these protocols are susceptible to *off-line dictionary attacks*: Many users choose passwords of relatively low entropy, so it is possible for the adversary to compile a dictionary of possible passwords.<sup>5</sup> Obviously, we can't prevent the adversary from trying all the passwords on-line, but such an attack can be made infeasible by simply placing a limit on the number of unsuccessful authentication attempts. On the other hand, an off-line search through the dictionary is quite doable. Here is an example of an off-line dictionary attack against a simple challenge-response protocol: The adversary overhears a challenge  $R$  and the associated response  $f(P, R)$  that involves the password. Now she can go off-line and run through all the passwords  $P'$  from a dictionary of likely passwords, comparing the value  $f(P', R)$  with  $f(P, R)$ . If one of the values matches the response, then the true password has been discovered.

A decade ago, Lomas et.al. [65] presented the first protocols which were resistant to these types of off-line dictionary attacks. The protocols assumed that the client had the server's public key and thus were not strictly password-only protocols. Gong et.al. [50] later presented their versions of these types of protocols. Recently, Halevi and Krawczyk [51] presented protocols and formal proofs of security for the same scenario as addressed by [65], i.e., where the client authenticates with a password and the server with a public key. Boyarsky [19] has addressed some problems with the Halevi-Krawczyk protocols in the multi-user scenario.

The EKE protocol [13] was the first password-authenticated key exchange protocol that did not require the user to know the server's public key. The idea of EKE was to use the password to symmetrically encrypt the protocol messages of a standard key

---

<sup>4</sup>Hashing the password does not offer more protection.

<sup>5</sup>See, for example, the experiment performed by Wu [101]. Briefly, using an off-line dictionary attack, he was able to discover 2045 passwords in a realm of slightly over twenty-five thousand users by verifying about 100 million candidate passwords.

exchange (e.g., Diffie-Hellman [35]). Then an attacker making a password guess could decrypt the symmetric encryption, but could not break the asymmetric encryption in the messages, and thus could not verify the guess. Following EKE, many protocols for password-authenticated key exchange were proposed which did not require the user to know the server's public key [14, 50, 49, 95, 56, 57, 66, 100]. Some of these protocols were, in addition, designed to protect against server compromise, so that an attacker that was able to steal data from a server could not later masquerade as a user without having performed a dictionary attack.<sup>6</sup> All of these protocol proposals contained informal arguments for security. However, the fact that some of these protocols were subsequently shown to be insecure [82, 83] should emphasize the importance of formal proofs of security.

### 3.2.3 Models for Secure Authentication and Key Exchange

Bellare and Rogaway [8] present the first formal model of security for entity authentication and key exchange, for the symmetric two party case. In [11] they extend it to the three party case. Blake-Wilson et.al. [16] further extend the model to cover the asymmetric setting. Independently, [67] and [7] present extensions to the model to allow for password authentication. Halevi and Krawczyk [51] and Boyarsky [19] present models which include both passwords and asymmetric keys (since both of those papers deal with protocols that are password-based, but rely on server public keys).

Bellare, Canetti, and Krawczyk [5] present a different model for security of entity authentication and key exchange, based on the multi-party simulatability tradition [3]. Shoup [93] refines and extends their model. We present a further extension of [93] that includes password authentication.

---

<sup>6</sup>Naturally, given the data from a server, an attacker could perform an off-line dictionary attack, since the server must know something that would allow verification of a user's password.

### 3.3 Model

For our proofs, we extend the formal notion of security for key exchange protocols from Shoup [93] to password-authenticated key exchange. We will prove security against a static adversary, i.e., one whose choice of whom to corrupt is independent of its view while attacking the protocol. We assume the adversary totally controls the network, a la [8].

Security for key exchange in [93] is defined using an ideal system, which describes the service (of key exchange) that is to be provided, and a real system, which describes the world in which the protocol participants and adversaries work. The ideal system should be defined such that an “ideal world adversary” cannot (by definition) break the security. Then, intuitively, a proof of security would show that anything an adversary can do in the real system can also be done in the ideal system, and thus it would follow that the protocol is secure in the real system.

#### 3.3.1 Definition of Security

The definition of security for key exchange given in [93] requires

1. **completeness:** for any real world adversary that faithfully delivers messages between two user instances with complimentary roles and identities, both user instances accept; and
2. **simulatability:** for every efficient real world adversary  $\mathcal{A}$ , there exists an efficient ideal world adversary  $\mathcal{A}^*$  such that  $RealWorld(\mathcal{A})$  and  $IdealWorld(\mathcal{A}^*)$  are computationally indistinguishable (here  $RealWorld(\cdot)$  and  $IdealWorld(\cdot)$  refer to real and ideal world transcripts, respectively, and are defined in Sections 3.3.3 and 3.3.2).

We will use this definition for password-authenticated key exchange as well, with no modifications. We can do this because, as will be seen below, our ideal model includes passwords explicitly. If it did not, we would have to somehow explicitly

state the probability of distinguishing real world from ideal world transcripts, given how many impersonation attempts the real world adversary made.

We now proceed with a description of the Ideal System for password authentication. The parts that are not directly related to passwords are taken from [93], except for a slight modification to handle mutual authentication.

### 3.3.2 Ideal System

We assume there is a set of (honest) *users*, indexed  $i = 1, 2, \dots$ . Each user  $i$  may have several *instances*  $j = 1, 2, \dots$ . Then  $(i, j)$  refers to a given *user instance*. A user instance  $(i, j)$  is told the identity of its partner, i.e., the user it is supposed to connect to (or receive a connection from). An instance is also told its *role* in the session, i.e., whether it is going to *open* itself for connection, or whether it is going to *connect* to another instance.

There is also an *adversary* that may perform certain operations, and a *ring master* that handles these operations by generating certain random variables and enforcing certain global consistency constraints. Some operations result in a record being placed in a *transcript*.

The ring master keeps track of session keys  $\{K_{ij}\}$  that are set up among user instances (as will be explained below, the key of an instance is set when that instance starts a session). In addition, the ring master has access to a random bit string  $R$  of some agreed-upon length (this string is not revealed to the adversary). We will refer to  $R$  as *the environment*. The purpose of the environment is to model information shared by users in higher-level protocols.

Since we deal with password authentication, and because passwords are not cryptographically secure, our system must somehow allow a non-negligible probability of an adversary successfully impersonating an honest user. We do this by including passwords explicitly in our model. We let  $\pi$  denote the function assigning passwords to pairs of users. To simplify notation, we will write  $\pi[A, B]$  to mean  $\pi[\{A, B\}]$  (i.e.,  $\pi[A, B]$  is by definition equivalent to  $\pi[B, A]$ ).

The adversary may perform the following types of operations:

**initialize user** [Transcript: ("initialize user",  $i, ID_i$ )]

The adversary assigns identity string  $ID_i$  to (new) user  $i$ . In addition, a random password  $\pi[ID_i, ID_{i'}]$  is chosen by the ring master for each existing user  $i'$ . The passwords are not placed in the transcript. This models the out-of-band communication required to set up passwords between users.

**set password** [Transcript: ("set password",  $i, ID', \pi$ )]

The identity  $ID'$  is required to be new, i.e., not assigned to any user. This sets  $\pi[ID_i, ID']$  to  $\pi$  and places a record in the transcript.

After  $ID'$  has been specified in a *set password* operation, it cannot be used in a subsequent *initialize user* operation.

**initialize user instance** [Transcript: ("init. user inst.",  $i, j, \text{role}(i, j), PID_{ij}$ )]

The adversary assigns a user instance  $(i, j)$  a role (one of  $\{\text{open}, \text{connect}\}$ ) and a user  $PID_{ij}$  that is supposed to be its partner. If  $PID_{ij}$  is not set to an identity of an initialized user, then we require that a *set password* operation has been previously performed for  $i$  and  $PID_{ij}$  (and hence there can be no future *initialize user* operation with  $PID_{ij}$  as the user ID).

**terminate user instance** [Transcript: ("terminate user instance",  $i, j$ )]

The adversary specifies a (previously initialized) user instance  $(i, j)$  to terminate.

**test instance password**

This is called with an instance  $(i, j)$  and a password guess  $\pi$ . The adversary queries if  $\pi = \pi[ID_i, PID_{ij}]$ . If this is true, the query is called a *successful guess on  $\{ID_i, PID_{ij}\}$*  (note that a successful guess on  $\{A, B\}$  is also a successful guess on  $\{B, A\}$ ).

This query may only be asked once per user instance. The instance has to be initialized and not yet engaged in a session (i.e., no *start session* operation has been performed for that instance). Note that the adversary is allowed to ask a *test instance password* query on an instance that has been terminated.

|  |   |
|--|---|
| <p>1. <b>open for connection from</b> <math>(i', j')</math>. This requires that</p> <ul style="list-style-type: none"> <li>• <math>role(i, j)</math> is “open,”</li> <li>• <math>(i', j')</math> has been initialized and has not been terminated,</li> <li>• <math>role(i', j')</math> is “connect,”</li> <li>• <math>PID_{ij} = ID_{i'}</math>,</li> <li>• <math>PID_{i'j'} = ID_i</math>,</li> <li>• no other instance is open for connection from <math>(i', j')</math>, and</li> <li>• no <i>test instance password</i> operation has been performed on <math>(i, j)</math>.</li> </ul> <p>The ring master generates <math>K_{ij}</math> randomly. We now say that <math>(i, j)</math> is open for connection from <math>(i', j')</math>.</p> | <p>2. <b>connect to</b> <math>(i', j')</math>. This requires that</p> <ul style="list-style-type: none"> <li>• <math>role(i, j)</math> is “connect,”</li> <li>• <math>(i', j')</math> has been initialized and has not been terminated,</li> <li>• <math>role(i', j')</math> is “open,”</li> <li>• <math>PID_{ij} = ID_{i'}</math>,</li> <li>• <math>PID_{i'j'} = ID_i</math>,</li> <li>• <math>(i', j')</math> was open for connection from <math>(i, j)</math> after <math>(i, j)</math> was initialized, and</li> <li>• no <i>test instance password</i> operation has been performed on <math>(i, j)</math>.</li> </ul> <p>The ring master sets <math>K_{ij} = K_{i'j'}</math>. We now say that <math>(i', j')</math> is no longer open for connection.</p> |
| <p>3. <b>expose</b>. This requires that either <math>PID_{ij}</math> has not been assigned to an identity of an initialized user, or there has been a successful guess on <math>\{ID_i, PID_{ij}\}</math>. The ring master sets <math>K_{ij}</math> to the value specified by the adversary.</p>   |   |

Table 3.1: Valid connection assignments for the *start session* operation

This query does not leave any records in the transcript.

**start session** [Transcript: (“start session”,  $i, j$ )]

The adversary specifies that a session key  $K_{ij}$  for user instance  $(i, j)$  should be constructed. The adversary specifies which *connection assignment* should be used. There are three possible connection assignments, as shown in Table 3.1.

Note that the connection assignment is not recorded in the transcript.

**application** [Transcript: (“application”,  $f, f(R, \{K_{ij}\})$ )]

The adversary is allowed to obtain any information she wishes about the environment and the session keys. (This models leakage of session key information in a real protocol through the use of the key in, for example, encryptions of

messages.) The function  $f$  is specified by the adversary and is assumed to be efficiently computable.

**implementation** [Transcript: ("impl", *cmnt*)]

The adversary is allowed to put in an “implementation comment” which does not affect anything else in the ideal world. This will be needed for generating ideal world views that are equivalent to real world views, as will be discussed later.

For an adversary  $\mathcal{A}^*$ ,  $IdealWorld(\mathcal{A}^*)$  is the random variable denoting the transcript of the adversary’s operations.

**Discussion (general key exchange):** Because keys exchanged between two honest users are not transmitted during the *start session* operations, it should be clear that key exchange is completely secure in the ideal world. Naturally, *application* operations may reveal keys. This models the use of the keys in a higher-level protocol. What we require from a secure key exchange protocol is that even given some partial information about the keys, the adversary shouldn’t be able to do anything in the real world that he couldn’t do in the ideal world. In other words, if a higher-level protocol is secure (in some appropriate sense) in an ideal system with keys generated by the ring master, that protocol should also be secure if we use a secure key exchange scheme.

**Remarks on mutual authentication:** As may be seen from the definition, the guarantees provided to the instances in *open* and *connect* roles are not completely symmetric. Specifically, an instance in a *connect* role is guaranteed that a connection will be established (since its partner is ready to accept the connection). On the other hand, an instance  $(i, j)$  in an *open* role has no guarantee that anyone will ever connect to it. All that is required is that there must exist an instance  $(i', j')$  of the partner that has not yet been terminated, and the only connection that either  $(i, j)$  or  $(i', j')$  could ever establish would be to each other. It appears impossible to completely eliminate the asymmetry of the definition without unrealistic assumptions.

The unilateral-authentication model of Shoup [93] is different from the one presented above in the following way: The *open* connection assignment does not specify the instance from which the connection is expected, and any instance of the partner is allowed to connect. On the other hand, the *connect* connection assignment still specifies an instance. It is unclear to us as to why one would need a model that provides authentication to only one of the parties. In any case, as Shoup remarks, authentication is not an important issue in ordinary key exchange. We will see, however, that authentication is more of an issue with passwords. This will become apparent when we consider the password key exchange model with implicit authentication in Section 3.5.1.

**Discussion (password authentication):** The major difficulty in designing an ideal system for password-authenticated key exchange is that there may be a non-negligible probability of an adversary guessing a password and impersonating a user. Thus either the definition of security must allow for a non-negligible simulation error, or the ideal system is forced to have some notion of passwords built in.

Our ideal system for password-authenticated key exchange explicitly uses (ring master generated) passwords. This forces any proof of security to have a simulator that has to be not only aware of a password-guessing attempt, but of exactly which password is being guessed. (In the proof for the PAK protocol presented in this thesis, the simulator is able to learn this information.) If this is not possible, then constructing and using a different ideal system would be required.

We did not specify how the ring master chooses passwords for pairs of users. The simplest model would be to have a dictionary  $\mathcal{D}$ , which is a set of strings, and let all passwords be chosen uniformly and independently from that dictionary. To achieve the strongest notion of security, though, we can give the adversary all the power, and simply let her specify the distribution of the passwords as an argument to the *initialize user* operation (the specification of the distribution would be recorded in the transcript). The passwords of a user could even be dependent on the passwords of other users. We note that our proofs of security do not rely on any specific distribution

of passwords, and would thus be correct even in the stronger model.

Why does our ideal system correctly describe the ideal world of password-authenticated key exchange? If two users successfully complete a key exchange, then the adversary cannot obtain the key or the password. This is modeled by the adversary not being allowed any *test instance password* queries for a successful key exchange. On the other hand, the adversary is allowed one test password query for any other key exchange, with successful impersonation only allowed if the adversary actually guesses the password correctly. This corresponds to an on-line impersonation/password-guessing attempt by the adversary. (One may think of this as modeling an adversary who attempts to log in to a server by sending a guessed password.)

We also model the ability for an adversary to set up passwords between any users and himself, using the *set password* query. This can be thought of as letting the adversary set up rogue accounts on any computer she wishes, as long as those rogue accounts have different user IDs from all the valid users.

### 3.3.3 Real System with Passwords

We now describe the real system in which we assume a password-authenticated key exchange protocol runs. Again, this is basically from [93], except that we do not concern ourselves with public keys and certification authorities, since all authentication is performed using shared passwords. Note that the same real system is used, no matter what authentication model we choose for our ideal system.

Users and user instances are denoted as in the ideal system. User instances are defined as state machines with implicit access to the user's  $ID$ ,  $PID$ , and password (i.e., user instance  $(i, j)$  is given access to  $\pi[ID_i, PID_{ij}]$ ). User instances also have access to private random inputs (i.e., they may be randomized). A user instance starts in some initial state, and may transform its state only when it receives a message. At that point it updates its state, generates a response message, and reports its status, either “**continue**”, “**accept**”, or “**reject**”, with the following meanings:

- “**continue**”: the user instance is prepared to receive another message.

- **“accept”**: the user instance (say  $(i, j)$ ) is finished and has generated a session key  $K_{ij}$ .
- **“reject”**: the user instance is finished, but has not generated a session key.

The adversary may perform the following types of operations:

**initialize user** [Transcript: ("initialize user",  $i, ID_i$ )]

As in the ideal system (with passwords), the adversary assigns identity string  $ID_i$  to (new) user  $i$  (as before). In addition, a random password  $\pi[ID_i, ID_{i'}]$  is chosen by the ring master for each existing user  $i'$ . The passwords are not placed in the transcript. This models the out-of-band communication required to set up passwords between users. As mentioned above with respect to the ideal system, the distribution of passwords may be fixed, or may be specified by the adversary as an argument to the operation (in the latter case, a description of the distribution would be recorded in the transcript).

**initialize user instance** [Transcript: ("init. user inst.",  $i, j, role(i, j), PID_{ij}$ )]

As in the ideal system, the adversary assigns a user instance  $(i, j)$  a role (one of  $\{\text{open, connect}\}$ ) and a user  $PID_{ij}$  that is supposed to be the partner of  $(i, j)$ .

**deliver message** [Transcript: ("impl", "message",  $i, j, InMsg, OutMsg, status$ )]

The adversary delivers  $InMsg$  to user instance  $(i, j)$ . The user instance updates its state, and replies with  $OutMsg$  and reports  $status$ . If  $status$  is “accept”, the record ("start session",  $i, j$ ) is added to the transcript, and if  $status$  is “reject”, the record ("terminate instance",  $i, j$ ) is added to the transcript.

**set password** [Transcript: ("set password",  $i, ID', \pi$ )]

As in the ideal system, this sets  $\pi[ID_i, ID']$  to  $\pi$  and places a record in the transcript.

**application** [Transcript: ("application",  $f, f(R, \{K_{ij}\})$ )]

As in the ideal system, the adversary is allowed to obtain any information she wishes about the environment and the session keys, except that the keys are now actual session keys generated by user instances.

**random oracle** [Transcript: ("impl", "random oracle",  $i, x, H_i(x)$ )]

The adversary queries random oracle  $i$  on a binary string  $x$  and receives the result of the random oracle query  $H_i(x)$ . Note that we do not allow *application* operations to query random oracles  $H_i$ . In other words, we do not give higher-level protocols access to the random oracles used by the key exchange scheme. (Although a higher-level protocol could have its own random oracle.) The adversary, however, does have access to all the random oracles.

For an adversary  $\mathcal{A}$ ,  $RealWorld(\mathcal{A})$  denotes the transcript of the adversary's operations. In addition to records made by the operations, the transcript will include the random coins of the adversary in an *implementation* record ("impl", "coins",  $coins$ ).

## 3.4 Explicit Authentication: The PAK Protocol

### 3.4.1 Preliminaries

Let  $\kappa$  and  $\ell$  denote our security parameters, where  $\kappa$  is the “main” security parameter and can be thought of as a general security parameter for hash functions and secret keys (say 128 or 160 bits), and  $\ell > \kappa$  can be thought of as a security parameter for discrete-log-based public keys (say 1024 or 2048 bits). Let  $\{0, 1\}^*$  denote the set of finite binary strings and  $\{0, 1\}^n$  the set of binary strings of length  $n$ . A real-valued function  $\epsilon(n)$  is *negligible* if for every  $c > 0$ , there exists  $n_c > 0$  such that  $\epsilon(n) < 1/n^c$  for all  $n > n_c$ .

Let  $q$  of size at least  $\kappa$  and  $p$  of size  $\ell$  be primes such that  $p = rq + 1$  for some value  $r$  co-prime to  $q$ . Let  $g$  be a generator of a subgroup of  $Z_p^*$  of size  $q$ . Call this subgroup  $G_{p,q}$ . We will often omit “mod  $p$ ” from expressions when it is obvious that we are working in  $Z_p^*$ .

Let  $\text{DH}(X, Y)$  denote the Diffie-Hellman value  $g^{xy}$  of  $X = g^x$  and  $Y = g^y$ . We assume the hardness of the *Decision Diffie-Hellman problem* (DDH) in  $G_{p,q}$ . One formulation is that given  $g, X, Y, Z$  in  $G_{p,q}$ , where  $X = g^x$  and  $Y = g^y$  are chosen randomly, and  $Z$  is either  $\text{DH}(X, Y)$  or random, each with half probability, determine

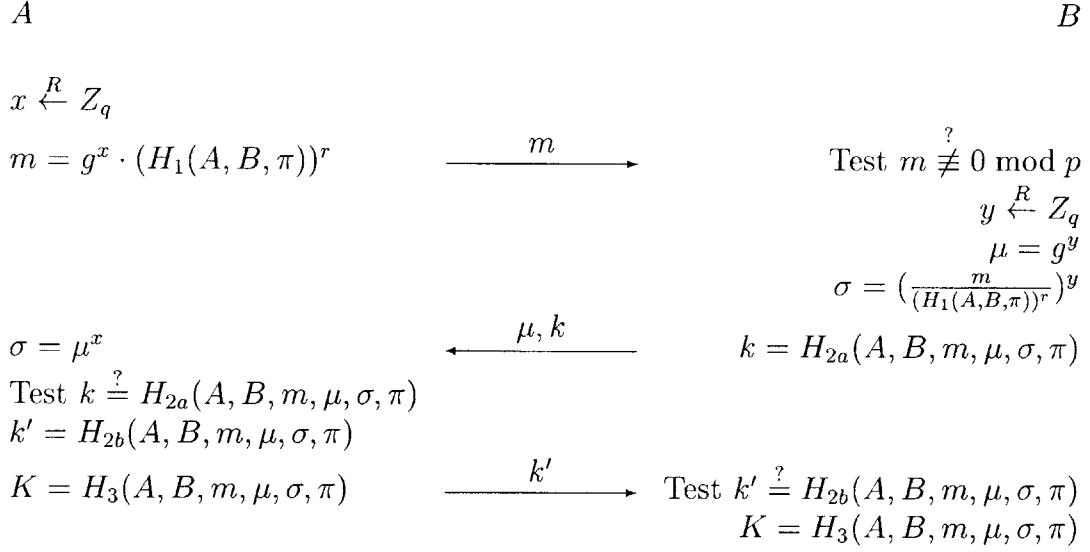


Figure 3-1: The PAK protocol, with  $\pi = \pi[A, B]$ . The resulting session key is  $K$ . If a “Test” returns false, the protocol is aborted.

if  $Z = \text{DH}(X, Y)$ . Breaking DDH implies a constructing a polynomial-time adversary that distinguishes  $Z = \text{DH}(X, Y)$  from a random  $Z$  with non-negligible advantage over a random guess.

### 3.4.2 The Protocol

Define hash functions  $H_{2a}, H_{2b}, H_3 : \{0, 1\}^* \rightarrow \{0, 1\}^\kappa$  and  $H_1 : \{0, 1\}^* \rightarrow \{0, 1\}^\eta$  (where  $\eta \geq \ell + \kappa$ ). We will assume that  $H_1$ ,  $H_{2a}$ ,  $H_{2b}$ , and  $H_3$  are independent random functions. Note that while  $H_1$  is described as returning a bit string, we will operate on its output as a number modulo  $p$ .

The PAK protocol is given in Figure 3-1.

**Theorem 6.** *The PAK protocol is a secure password-authenticated key exchange protocol in the explicit-authentication model.*

The proof is given in Section 3.7.

## 3.5 Implicit Authentication: The PPK Protocol

We first describe an Ideal System with Implicit Authentication, and then describe the PPK protocol. Note that we still use the Real System from Section 3.3.3.

### 3.5.1 Ideal System with Implicit Authentication

Here we consider protocols in which the parties are *implicitly* authenticated, meaning that if one of the communicating parties is not who it claims to be, it simply won't be able to obtain the session key of the honest party. However, the honest party (which could be playing the role of "open" or "connect") would still open the session, but with no one able to actually communicate with on that session.<sup>7</sup>

Thus some of the connections may be “dangling.” We will allow two new connection assignments:

**dangling open.** This requires  $role(i, j)$  to be “open.”

**dangling connect.** This requires  $role(i, j)$  to be “connect.”

In both cases, the ring master generates  $K_{ij}$  randomly.

To use implicit authentication with passwords, we will make the following rules:

- We no longer require that no two instances are open for connection from the same instance (since for implicit authentication we don't care that every accepting instance has a unique partner instance).
- Dangling connection assignments are allowed even for instances on which the *test instance password* query has been performed.
- A *test instance password* query is allowed on an instance that has started a session with a dangling connection assignment. In that case, if the test is successful, then the ring master returns to the adversary the key of the instance (the adversary is “rewarded” with the key, since the usual “reward,” i.e., the

---

<sup>7</sup>In a later version of [93], Shoup also deals with implicit authentication, but in a different way. We feel our solution is more straightforward and intuitive.

ability to make an *expose* connection assignment, is not usable once the dangling connection assignment has been made).

We still restrict the number of *test instance password* queries to at most one per instance. The rules relating to other connection assignments do not change.

The reason for the permissiveness in *test instance password* is that an instance with a dangling connection assignment can't be sure that it wasn't talking to the adversary. All that is guaranteed is that the adversary won't be able to get the key of that instance, unless she correctly guesses the password.

In practice, this means that we can't rule out an unsuccessful password guess attempt on an instance until we can confirm that some partner instance has obtained the same key. (If another instance has indeed obtained the same key, that guarantees that a non-dangling connection assignment was made, since otherwise the keys would be independent.) It follows that if we are trying to count the number of unsuccessful login attempts (e.g., so that we can lock the account when some threshold is reached), we can't consider an attempt successful until we get some kind of confirmation that the other side has obtained the same key. We thus see that key confirmation (which, in our model, is equivalent to explicit authentication) is indeed relevant when we use passwords.

### 3.5.2 PPK Protocol

If we don't require explicit authentication, we can make a much more efficient protocol. The PPK protocol requires only two rounds of communication. The protocol is given in Figure 3-2. Here  $H'_1 : \{0, 1\}^* \rightarrow \{0, 1\}^n$  is another random function. PPK has the same basic structure as PAK, except without the “authentication” values  $k$  and  $k'$ .

Note that in PPK, as opposed to PAK, both the  $m$  and  $\mu$  values need to be “encrypted” with the password (by multiplication with a random oracle value). The reason is that in PPK these values are not in any way authenticated. If, as in PAK,  $\mu$  wasn't “encrypted,” then an adversary could carry out the following attack:



## 3.6 Resilience to Server Compromise: The PAK-X Protocol

### 3.6.1 Ideal System with Passwords: Resilience to Server Compromise

We now define a system in which one party is designated as a server, and which describes the ability of an adversary to obtain information about passwords stored on the server, along with the resultant security. To accomplish this, one role (open or connect) is designated as the *server* role, while the other is designated as the *client* role.

We add the *test password* and *get verifier* operations, and change the *start session* operation.

#### **test password**

This query takes two users, say  $i$  and  $i'$ , as arguments, along with a password guess  $\pi$ . If a *get verifier* query has been made on  $\{i, i'\}$ , then this returns whether  $\pi = \pi[ID_i, ID_{i'}]$ . If the comparison returns true, this is called a successful guess on  $\{ID_i, ID_{i'}\}$ . If no *get verifier* has been made on  $\{i, i'\}$ , then no answer is returned (but see the description of *get verifier* below).

This query does not place a record in the transcript. It can be asked any number of times, as long as the next query after every *test password* is of type *implementation*. (The idea of the last requirement is that a *test password* query has to be caused by a “real-world” operation, which leaves an *implementation* record in the transcript.)

#### **get verifier** [Transcript: (“get verifier”, $i, i'$ )]

Arguments: users  $i$  and  $i'$ . For each *test password* query on  $\{i, i'\}$  that has previously been asked (if any), returns whether or not it was successful. If any one of them actually was successful, then this *get verifier* query is called a successful guess on  $\{ID_i, ID_{i'}\}$ . Note that the information about the success of

failure of *test password* queries is *not* placed in the transcript.

**start session** [Transcript: ("start session",  $i, j$ )]

In addition to the rules specified previously, a connection assignment of *expose* for client instance  $(i, j)$  is allowed at any point after a *get verifier* query on users  $i$  or  $i'$  has been performed, where  $ID_{i'} = PID_{ij}$ .

The *test password* query does not affect the legality of *open* and *connect* connection assignments.

### 3.6.2 Real System: Resilience to Server Compromise

In a real system that has any resilience to server compromise, the server must not store the plaintext password. Instead, the server stores a *verifier* to verify a user's password. Thus the protocol has to specify a PPT verifier generation algorithm  $VGen$  that, given a set of user identities  $\{A, B\}$ , and a password  $\pi$ , produces a verifier  $V$ .

As above for  $\pi[A, B]$ , we will write  $V[A, B]$  to mean  $V[\{A, B\}]$ .

A user instance  $(i, j)$  in the server role is given access to  $V[ID_i, PID_{ij}]$ . A user instance  $(i, j)$  in the client role is given access to  $\pi[ID_i, PID_{ij}]$ .

The changes to the *initialize user* and *set password* operations are given here:

**initialize user** [Transcript: ("initialize user",  $i, ID_i$ )]

In addition to what is done in the basic real system,  $V[ID_i, ID_{i'}] = VGen(\{ID_i, ID_{i'}\}, \pi[ID_i, ID_{i'}])$  is computed for each  $i'$ .

**set password** [Transcript: ("set password",  $i, ID', \pi$ )]

In addition to what is done in basic real system,  $V[ID_i, ID']$  is set to  $VGen(\{ID_i, ID'\}, \pi)$ .

We add the *get verifier* operation here:

**get verifier** [Transcript: ("get verifier",  $i, i'$ ), followed by ("impl", "verifier",  $i, i'$ ,  $V[ID_i, ID_{i'}]$ )]

The adversary performs this query with  $i$  and  $i'$  as arguments, with  $V[ID_i, ID_{i'}]$  being returned.

### 3.6.3 PAK-X Protocol

In our protocol, we will designate the *open* role as the client role. We will use  $A$  and  $B$  to denote the identities of the client and the server, respectively. In addition to the random oracles we have used before, we will use additional functions  $H_0 : \{0, 1\}^* \rightarrow \{0, 1\}^{|q|+\kappa}$  and  $H'_0 : \{0, 1\}^* \rightarrow \{0, 1\}^{|q|+\kappa}$ , which we will assume to be random functions. The verifier generation algorithm is

$$VGen(\{A, B\}, \pi) = g^{v[A, B]},$$

where we define  $v[A, B] = H_0(\min(A, B), \max(A, B), \pi)$  (we need to order user identities, just so that any pair of users has a unique verifier).

The PAK-X protocol is given in Figure 3-3.

**Theorem 8.** *The PAK-X protocol is a secure password-authenticated key exchange protocol in the explicit-authentication model, with resilience to server compromise.*

The completeness requirement follows directly by inspection. The proof of simulatability appears in Section 3.9. The basic structure of the proof is the same as for the PAK protocol. A major technical difficulty in the simulation is the case when the adversary has obtained the verifier, and is now acting as the server (in fact, the security of the SNAP-X protocol [67] has only been shown under the assumption that such a scenario does not occur). The difficulty is that the simulator needs to verify the value of  $V^{H'_0(A, B, c)}$  without knowing  $v$ . This is achieved by checking all the  $H'_0$  queries until we find a query  $c$  that gives  $a = g^{H'_0(A, B, c)}$ . We can then use this value of  $c$  to determine the correct value of  $V^{H'_0(A, B, c)}$ . (A similar technique was used independently in [44] to obtain an efficient encryption scheme secure against an adaptive chosen-ciphertext attack.)

## 3.7 Security of the PAK Protocol

The completeness requirement follows directly by inspection. Here we prove that the simulatability requirement holds. The basic technique is essentially that of Shoup

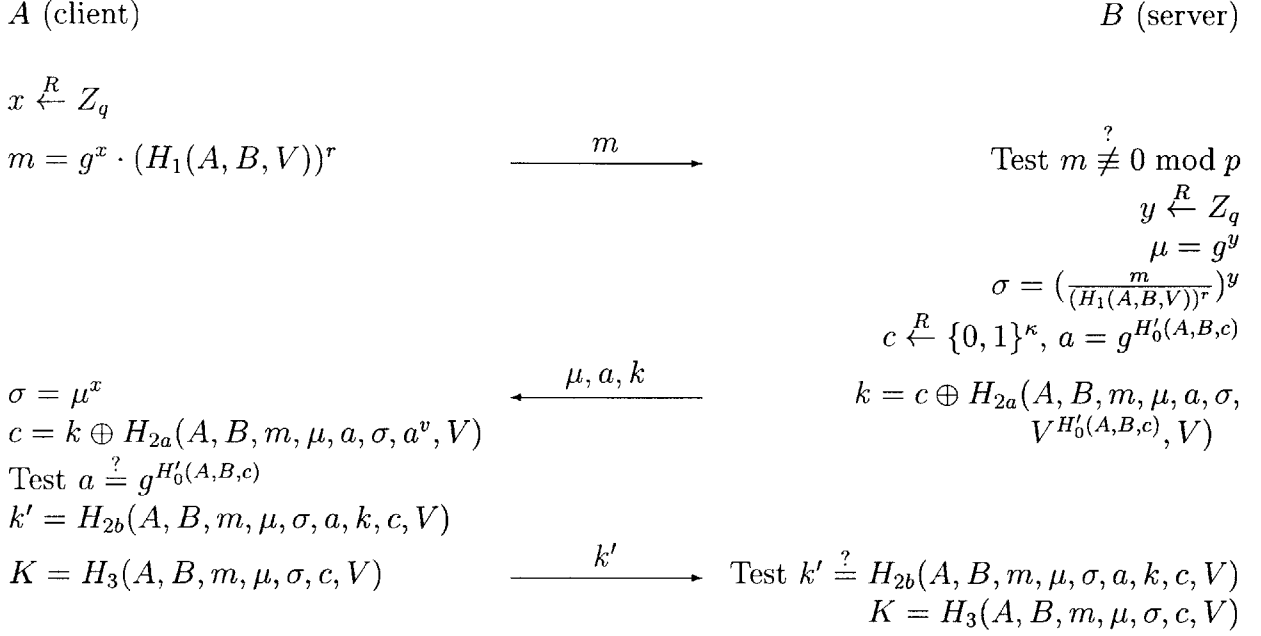


Figure 3-3: The PAK-X protocol, with  $\pi = \pi[A, B]$ ,  $v = v[A, B]$ , and  $V = V[A, B]$ . The resulting session key is  $K$ .

[93]. The idea is to create an ideal world adversary  $\mathcal{A}^*$  by running the real world adversary  $\mathcal{A}$  against a simulated real system, which is built on top of the underlying ideal system. In particular,  $\mathcal{A}^*$  (i.e., the simulator combined with  $\mathcal{A}$ ) will behave in the ideal world just like  $\mathcal{A}$  behaves in the real world, except that idealized session keys will be used in the real world simulation instead of the actual session keys computed in the real system.

Thus our proof consists of constructing a simulator (that is built on top of an ideal system) for a real system so that the transcript of an adversary attacking the simulator is computationally indistinguishable from the transcript of an adversary attacking the real system.

Let  $T$  be the running time of the adversary. (We will also use this as a bound on the number of operations the adversary performs in the real system.)  $T$  must be polynomial in the security parameter  $\kappa$ . W.o.p. stands for “with overwhelming probability,” i.e., with probability at least  $1 - \epsilon$ , for some  $\epsilon$  which is negligible in the security parameter  $\kappa$ .

Let  $\text{DH}(X, Y)$  denote the Diffie-Hellman value  $g^{xy}$  of  $X = g^x$  and  $Y = g^y$ .

### 3.7.1 The Simulator

The general idea of our simulator is to try to detect guesses on the password (by examining the adversary’s random oracle queries) and turn them into *test instance password* queries. If the simulator does not notice a password guess, then it either sets up a connection between two instances (if all the messages between them have been correctly relayed), or rejects (otherwise).

The main difficulty in constructing the simulator is that we need to simulate the protocol without knowing the actual passwords. We solve this problem as follows: It may be seen that the password only appears in the protocol as an argument to random oracle queries. Now, whenever the actual protocol would use the result of a random oracle query whose arguments involve the password, we will simply substitute a random value for the oracle’s response. We can think of this as an “implicit” oracle call, i.e., one where we know the value returned, even though we don’t (as of yet, at least) know the arguments. In handling the adversary’s explicit random oracle queries, as well as those protocol operations that use random oracles, we need to make sure that we don’t use inconsistent values for the result of a random oracle on a certain input. In particular, we need to be able to detect if an adversary’s query to a random oracle might match a prior implicit oracle call. We will say that an oracle query is *shadowed* by an implicit oracle query if the two queries would be equal for some feasible (i.e., not yet ruled out) value of the password.

Indeed, one of the main concerns in the proof is dealing with the possible shadowings: either detecting them when they occur, or showing that they are impossible. The possible shadowings in the PAK simulator are shown in Table 3.2.

In the process of describing the simulator, we will show that the transcript of the simulation in the ideal world is computationally indistinguishable from the transcript of the actual adversary in the real world, step by step, unless it is possible to construct an algorithm to break the Decision Diffie-Hellman problem.

Let us now proceed with the technical details. Say an *initiator* is an instance that

|          | Current  |          |         |            |            |            |
|----------|----------|----------|---------|------------|------------|------------|
| Earlier  | $H_{2a}$ | $H_{2b}$ | $H_3$   | B1         | B1'        | A2         |
| $H_{2a}$ |          |          |         | $\epsilon$ | $\epsilon$ |            |
| $H_{2b}$ |          |          |         |            |            | Claim 3    |
| $H_3$    |          |          |         |            |            | Claim 3    |
| B1       | Claim 3  |          |         | $\epsilon$ | $\epsilon$ |            |
| B1'      | Explicit |          |         | $\epsilon$ | $\epsilon$ |            |
| A2       |          | Claim 3  | Claim 3 |            |            | $\epsilon$ |

The names of rows and columns refer to parts of the simulator: handling of the adversary's random oracle queries ( $H_i$ ) and message responses (e.g., B1). A message-response action without a prime (e.g., B1) denotes that there is a matching conversation with an acceptable partner, while a prime (e.g., B1') denotes the lack of one. Only actions that could possibly cause shadowing are shown in the table.

The columns of the table correspond to an action being currently handled by the simulator. The rows correspond to earlier actions. An entry in the table indicates whether or not while handling the current action we need to be concerned for shadowing caused by an earlier action.

The entries of the table have the following meaning:

**blank** shadowing is impossible by the structure of the simulator,

$\epsilon$  shadowing is possible, but has a negligible probability (statistically),

**claim** if the adversary can cause such shadowing with nonnegligible probability, then we can construct a DDH distinguisher (as shown in the referenced claim),

**explicit** shadowing can indeed occur, and is explicitly dealt with in the simulator.

Table 3.2: Possible shadowings in the PAK simulator.

sends the first message in the protocol, and a *responder* is an instance that sends the second message in the protocol. We will always write  $(i, j)$  for the user instance that is an initiator, and  $(i', j')$  for the user instance that is a responder. Let  $A$  (resp.  $B$ ) be the ID of the current initiator (resp. responder), i.e., either  $ID_i$  or  $PID_{i'j'}$  (resp.  $PID_{ij}$  or  $ID_{i'}$ ), depending on the context. Let  $\pi^* = \pi[A, B]$ . An *acceptable partner* for an instance  $(i, j)$  (resp.  $(i', j')$ ) is any instance  $(i', j')$  (resp.  $(i, j)$ ) with  $PID_{i'j'} = ID_i$  and  $PID_{ij} = ID_{i'}$  (resp.  $PID_{ij} = ID_{i'}$  and  $PID_{i'j'} = ID_i$ ). We say that two instances had a *matching conversation* if all the messages sent by one were received by the other (preserving order) and vice versa. (This definition is as in [8], except that we don't care about the timings, i.e., we don't rule out the possibility of a message being received before it is sent. However, we will see that the probability of this is negligible in our protocol. Also note that, as opposed to [8], matching conversations are not used in our definition of security, but only as a notion to better illustrate our proof.) When we say that an instance in the open role has had a matching conversation with an instance in the connect role, this does not make any requirements on the last message (i.e., the last message is not required to have been delivered properly). However, in order for an instance in the connect role to have a matching conversation with an instance in the open role, all the messages need to be delivered properly.

We now describe the actions of the simulator for each possible operation of the adversary in the real protocol. An *initialize user instance* or an *application* operation is simply passed on to the ideal system. A *set password* operation is also passed through to the ideal system (and the password is recorded by the simulator). A *deliver message* operation is dealt with depending on the state of the user instance involved. This state includes the role of the user instance, and the previous messages to and from that user instance. A *random oracle* operation is answered depending on which random oracle is queried. The responses to *deliver message* and *random oracle* operations are specified below.

We name the actions of user instances on *deliver message* operations as follows:

**A0** Initiator instance action to start the protocol (i.e., an A0 action for an initiator instance is generating and sending  $m$ ).

**B1** Responder instance action upon receiving the first message (i.e.,  $m$ ).

**A2** Initiator instance action upon receiving the message from the responder (i.e.,  $(\mu, k)$ ).

**B3** Responder instance action upon receiving the second message from the initiator (i.e.,  $k'$ ).

For example,  $B1(m)$  denotes an adversary's *deliver message* operation on some responder instance, with  $InMsg$  being  $m$ .

We now describe some general rules for handling *deliver message* operations: We discard all improper messages to user instances, just as would be done in the real system. These messages may be improper because, for instance, they are not formatted correctly, or the user identities do not match. If the partner ID of a user instance is not set to an identity of an initialized user, then the original protocol is followed, and the *expose* connection assignment is used for a *start session* operation (this is possible, since the simulator has seen the necessary password in the *set password* operation).

Here are some general rules for handling *random oracle* operations: The simulator keeps a record of all random oracle query-response pairs (of course, this is only done for explicit oracle queries). If a query made to a random oracle matches a previous query to the same random oracle, the stored response is returned. If a random oracle is given user identities  $A$  and  $B$  as arguments, and either (or both) of  $A$  and  $B$  is not the identity of a valid user, then the query is answered with a random string (as in the real system).

Detailed descriptions of how the simulator responds to *deliver message* and *random oracle* operations (that do not fall under the above rules) follow:

1.  $H_1(A, B, \pi)$

Generate  $\alpha[A, B, \pi] \xleftarrow{R} Z_q$  and store it. Then generate  $h \xleftarrow{R} Z_p^*$  and  $\beta \xleftarrow{R} Z_{[2^n/p]}$ , and return  $(h^q g^{\alpha[A, B, \pi]} \bmod p) + \beta p$ . Note that this will be indistinguishable from

a random bit string of length  $\eta$ , since  $h^q g^{\alpha[A,B,\pi]} \bmod p$  is a random element from  $Z_p^{*8}$  and  $\frac{2^7 \bmod p}{2^\eta}$  is negligible.

Note that  $(H_1(A, B, \pi))^r = (h^q g^{\alpha[A,B,\pi]})^r = h^{qr} g^{r \cdot \alpha[A,B,\pi]} = g^{r \cdot \alpha[A,B,\pi]}$ .

## 2. $H_{2a}(A, B, m, \mu, \sigma, \pi)$

**Case 1** Some prior B1 query has recorded a tuple of the form  $(i', j', m, \mu, y, k_{i'j'})$ , with  $A = PID_{i'j'}$  and  $B = ID_{i'}$ , for some values of  $y$  and  $k_{i'j'}$ . This implies that a B1( $m$ ) query was made to  $(i', j')$  and returned  $(\mu, k_{i'j'})$ , but no A0 query to an acceptable partner returned  $m$ —see the B1 query description below. (In other words, this  $H_{2a}$  query might be shadowed by the implicit  $H_{2a}$  query from case 2 of the B1 action.)

W.o.p., there will be at most one such tuple, since  $\mu$  values stored in these tuples are random and independent. If  $H_1(A, B, \pi)$  has been asked, and  $(\frac{m}{(H_1(A,B,\pi))^r})^y = \sigma$ , then:

- (a) If there has been a successful guess on  $\{A, B\}$  and  $\pi$  was the password in that guess, call this  $H_{2a}$  query a *successful  $H_{2a}$  query on  $(i', j')$* .
- (b) If there hasn't been a successful guess on  $\{A, B\}$ , and there never was an unsuccessful guess on  $\{A, B\}$  for  $\pi$ :
  - i. If a *test instance password* operation has not previously been performed on  $(i', j')$ , perform a *test instance password* operation with arguments  $(i', j')$  and  $\pi$ . If the test is successful, we also call this query a *successful  $H_{2a}$  query on  $(i', j')$* .
  - ii. If a *test instance password* operation has previously been performed on  $(i', j')$ , then abort. We will call this event an  *$H_{2a}$  failure*.

Finally, if this query is a successful  $H_{2a}$  query on  $(i', j')$  for some  $(i', j')$ , then return  $k_{i'j'}$ . Otherwise, return  $k \xleftarrow{R} \{0, 1\}^\kappa$ .

---

<sup>8</sup>To see this, note that  $h^q$  is a random element from the subgroup of order  $r$  in  $Z_p^*$  and  $g^{\alpha[A,B,\pi]}$  is a random element of the subgroup of order  $q$  in  $Z_p^*$ .

If no  $H_{2a}$  failure occurs as a result of this  $H_{2a}$  query, then the simulation will be indistinguishable from the real world, as follows: In the real world the  $k$  value sent by  $(i', j')$  is  $H_{2a}(A, B, m, \mu, \text{DH}(\mu, \frac{m}{(H_1(A, B, \pi^*))^r}), \pi^*)$ . Therefore, if this  $H_{2a}$  query is successful, it will return a value consistent with the adversary's view. On the other hand, if this query is not successful, then generating a random response is indistinguishable from the adversary's view in the real world.

The following claim shows that w.o.p.  $\mathcal{A}$  will not ask an  $H_{2a}$  query that causes an  $H_{2a}$  failure. (The  $H_{2b}$  case in the claim is necessary to handle B3 operations.)

**Claim 1.** *Let  $\mu$  be returned by a  $B1(m)$  query to  $(i', j')$ . Let  $A = \text{PID}_{i'j'}$  and  $B = \text{ID}_{i'}$ . Then w.o.p. it will not happen that the two oracle queries  $H_s(A, B, m, \mu, \text{DH}(\mu, \frac{m}{(H_1(A, B, \pi_1))^r}), \pi_1)$ , and  $H_t(A, B, m, \mu, \text{DH}(\mu, \frac{m}{(H_1(A, B, \pi_2))^r}), \pi_2)$  will be asked, with  $s, t \in \{2a, 2b\}$ , unless either  $\pi_1 = \pi_2$ , or by the time of the second query there has already been a successful guess on  $\{A, B\}$ .*

Note that in this claim we speak both about queries made by the adversary, and explicit queries made within the simulator (e.g., in case 2 of the A2 operation).

The proof appears in Section 3.7.2.

Note that an  $H_{2a}$  query can result in a *test instance password* query on  $(i', j')$  only if a tuple  $(i', j', m, \mu, y, k_{i'j'})$  has previously been recorded. It will become apparent below (see the B1 query description) that, w.o.p., this could only happen if  $(i', j')$  has not had a matching conversation with an acceptable partner. We have just proven one part of the following claim:

**Claim 2.** *If a test instance password query is made on a responder instance  $(i', j')$ , then  $(i', j')$  has not had a matching conversation with an acceptable partner.*

The other part is shown in case 3b of B3 (which is the only other place where a *test instance password* could be made on a responder instance).

**Case 2** No tuple of the form  $(i', j', m, \mu, y, k_{i'j'})$  has been recorded with  $A = PID_{i'j'}$  and  $B = ID_{i'}$ .

Return  $k \xleftarrow{R} \{0, 1\}^\kappa$ . The only way this response could be distinguishable from the real system is if this query would be shadowed by an implicit  $H_{2a}$  query from case 1 of some B1 action, i.e., if  $m$  and  $\mu$  are from a matching conversation of two valid user instances, and  $\sigma = \text{DH}(\mu, \frac{m}{(H_1(A, B, \pi))^r})$ . However, by the following claim, w.o.p. this will not occur.

**Claim 3.** *Let  $m$  be returned by an A0 query to  $(i, j)$  with  $A = ID_i$  and  $B = PID_{ij}$ , and  $\mu$  be returned by a subsequent B1( $m$ ) query to  $(i', j')$  with  $B = ID_{i'}$  and  $A = PID_{i'j'}$ . Then, w.o.p., there will never be (neither before nor after the A0 and B1 queries) an oracle query  $(A, B, m, \mu, \text{DH}(\mu, \frac{m}{(H_1(A, B, \pi))^r}), \pi)$  to  $H_{2a}$ ,  $H_{2b}$ , or  $H_3$ , for any  $\pi$ .*

The proof of the claim appears in Section 3.7.2. Note that we only consider B1( $m$ ) queries subsequent to an A0 query that returns  $m$ , since the probability of their occurrence prior to an A0 query that returns  $m$  is negligible (by randomness of  $m$ ).

3.  $H_{2b}(A, B, m, \mu, \sigma, \pi)$

Return  $k' \xleftarrow{R} \{0, 1\}^\kappa$ . This is indistinguishable from the real system by the following argument: The only implicit  $H_{2b}$  query that could shadow this one is the query from case 1 of an A2 action. However, that shadowing is, w.o.p., impossible by Claim 3. (Note that the  $H_{2b}$  query from case 2 of an A2 action is explicit, i.e., all of its arguments are known, and so a query-response pair would be stored for it.)

4.  $H_3(A, B, m, \mu, \sigma, \pi)$

Return  $K \xleftarrow{R} \{0, 1\}^\kappa$ . As for  $H_{2b}$  queries, indistinguishability easily follows by Claim 3.

5. A0 query to  $(i, j)$

Generate and store  $w \xleftarrow{R} Z_q$ , and send  $m = g^w$ . Clearly,  $m$  is uniformly drawn from  $G_{p,q}$ , just as in the real system.

6. B1( $m$ ) query to  $(i', j')$

Generate  $y \xleftarrow{R} Z_q$  and  $k_{i'j'} \xleftarrow{R} \{0, 1\}^\kappa$ . Send  $\mu = g^y$  and  $k = k_{i'j'}$ . Now consider two cases:

**Case 1** The value of  $m$  has been sent by an acceptable partner.

We can think of  $k_{i'j'}$  as the result of an implicit query

$$H_{2a}(A, B, m, \mu, \text{DH}(\mu, \frac{m}{(H_1(A, B, \pi^*))^r}), \pi^*).$$

Since  $\mu$  is a freshly chosen random number, this implicit query, w.o.p., will not shadow any prior  $H_{2a}$  queries (whether explicit or implicit). Consequently,  $k_{i'j'}$  will be indistinguishable from the value sent in the real system.

**Case 2** The value of  $m$  has not been sent by an acceptable partner.

In this case, record the tuple  $(i', j', m, \mu, y, k_{i'j'})$ . It is important to note (for Claim 2, that if this tuple is recorded, then, w.o.p.,  $(i', j')$  will never be considered to have a matching conversation with an acceptable partner, due to the fact that values of  $m$  are generated randomly by initiator instances. (Recall that in our definition of “matching conversation” we are not concerned with the timing of sends and receives.)

As in case 1, we can view  $k_{i'j'}$  as the result of an implicit query

$$H_{2a}(A, B, m, \mu, \text{DH}(\mu, \frac{m}{(H_1(A, B, \pi^*))^r}), \pi^*).$$

The value of  $k_{i'j'}$  will be indistinguishable from the one sent in the real system for the same reason as before.

7. A2( $\mu, k$ ) query to  $(i, j)$

**Case 1**  $(i, j)$  has had a matching conversation with an acceptable partner  $(i', j')$ .

Generate  $k'_{ij} \xleftarrow{R} \{0, 1\}^\kappa$ , send  $k' = k'_{ij}$ , set status to Accept, and perform a *start session* operation with the connection assignment *open for connection from*  $(i', j')$ . This corresponds to the following implicit oracle queries:

$$\begin{aligned} k'_{ij} &= H_{2b}(A, B, m, \mu, \text{DH}(\mu, \frac{m}{(H_1(A, B, \pi^*))^r}), \pi^*), \\ K_{ij} &= H_3(A, B, m, \mu, \text{DH}(\mu, \frac{m}{(H_1(A, B, \pi^*))^r}), \pi^*), \end{aligned}$$

where  $K_{ij}$  is the random session key assigned by the ring master at the time of the *start session* operation.

By Claim 3, these implicit queries couldn't shadow any past or future explicit queries. It is also easy to see that these implicit queries will not shadow any other implicit queries, unless there is a collision of  $m$  values between two initiator instances (and that only occurs with negligible probability).

The *open* connection assignment will be legal, since the only place that the simulator could perform a *test instance password* operation on  $(i, j)$  would be in case 2 of the A2 query (see there). Also, w.o.p., we will never have two initiator instances  $(i_1, j_1)$  and  $(i_2, j_2)$  that are open for connection from the same responder instance  $(i', j')$ , since that would imply that  $(i_1, j_1)$  and  $(i_2, j_2)$  generated the same  $m$  value (otherwise, they couldn't both have had matching conversations with  $(i', j')$ ).

**Case 2**  $(i, j)$  has not had a matching conversation with an acceptable partner.

Look for a value of  $\pi$  for which an  $H_1(A, B, \pi)$  query has been made, and another query  $H_{2a}(A, B, m, \mu, \mu^{w-r\alpha[A, B, \pi]}, \pi)$  has been made and returned  $k$ . (W.o.p. there will be at most one such value of  $\pi$ , due to the randomness of  $H_{2a}$ .) If no such  $\pi$  is found, then reject. This is indistinguishable from the real system, since w.o.p. the  $k$  received would be incorrect in the real

system (i.e., the correct  $k = H_{2a}(A, B, m, \mu, \text{DH}(\mu, \frac{m}{(H_1(A, B, \pi^*))^r}), \pi^*)$  would be independent of the adversary's view).

Otherwise (if such a  $\pi$  is found), perform a *test instance password* operation with arguments  $(i, j)$  and  $\pi$  (note that this is the only place where we could perform this operation for an initiator instance, and no session has been started yet, so the operation is legal). If the guess is not successful, then reject (this is indistinguishable from the real system, by an argument similar to the one above). If the guess is successful, then:

- (a) Set  $k' = H_{2b}(A, B, m, \mu, \mu^{w-r\alpha[A, B, \pi]}, \pi)$ .
- (b) Send  $k'$ .
- (c) Accept.
- (d) Set  $K = H_3(A, B, m, \mu, \mu^{w-r\alpha[A, B, \pi]}, \pi)$ .
- (e) Expose using session key  $K$  (this is allowed, since there was a successful guess on the password).

Note that the values of  $k'$  and  $K$  are computed through explicit oracle queries (we can think of these as subroutine calls within the simulator), and the queries and responses are recorded, as usual. It is clear that the values of  $k'$  and  $K$  produced in this case are the same as would be computed in the real system.

#### 8. B3( $k'$ ) query to $(i', j')$

**Case 1**  $(i', j')$  has had a matching conversation with an acceptable partner  $(i, j)$ .

Set status to Accept and perform a *start session* operation with a *connect to*  $(i, j)$  connection assignment. This is legal because

- (a) w.o.p., the instance  $(i, j)$  will still be open for connection by the randomness of  $\mu$  values sent by responder instances (so that, w.o.p., for each initiator  $(i, j)$ , there will be at most one  $(i', j')$  with which it has

had a matching conversation, and so at most one instance will try to connect to  $(i, j)$ ), and

- (b) by Claim 2, there could not have been a *test instance password* query on  $(i', j')$ , since  $(i', j')$  has had a matching conversation with an acceptable partner.

**Case 2** The current value  $m$  has been sent by some acceptable partner  $(i, j)$ , and  $\mu$  and  $k$  have been received by  $(i, j)$ , but the value of  $k'$  that was received has not been sent by  $(i, j)$ .

Reject. This is indistinguishable from the real system since  $k'$  is invalid.

**Case 3** Neither Case 1 nor Case 2 holds.

Look for a value of  $\pi$  such that an  $H_1(A, B, \pi)$  query has been asked, and a query  $H_{2b}(A, B, m, \mu, (\frac{m}{(H_1(A, B, \pi))^r})^y, \pi)$  has been asked and returned  $k'$ . (W.o.p. there will be at most one such value of  $\pi$ , due to the randomness of  $H_{2b}$ .) If no such  $\pi$  is found, then reject. This is indistinguishable from the real system, since w.o.p. the  $k'$  received would be incorrect in the real system (i.e., the correct  $k'$  would be independent of the adversary's view). (The only way  $k'$  could be correct, other than through a random guess, is if it is the result of an implicit  $H_{2b}$  query in case 1 of A2. However, it is easy to see that if that was the case, we couldn't get to case 3 of B3.)

Otherwise (if such a  $\pi$  is found), check if there was a successful guess on  $\{A, B\}$ .

**Case 3a** There was a successful guess on  $\{A, B\}$ . If that guess was not  $\pi$ , then reject.

**Case 3b** There was no successful guess on  $\{A, B\}$ . If there already was an unsuccessful guess on  $\{A, B\}$  for  $\pi$ , then reject. Otherwise, perform a *test instance password* operation with arguments  $(i', j')$  and  $\pi$ . (Note that in this case  $(i', j')$  has not had a matching conversation with an acceptable partner. This completes the proof of Claim 2.) If the guess is not successful, then reject.

We can easily see by Claim 1 (using  $s = 2a$  and  $t = 2b$ ), that, w.o.p., this procedure will not make a *test instance password* query on  $(i', j')$  if one has already been made before.

If we haven't rejected yet, then

- (a) set status to Accept,
- (b) set  $K = H_3(A, B, m, \mu, (\frac{m}{(H_1(A, B, \pi))^r})^y, \pi)$ , and
- (c) expose using session key  $K$  (this is allowed, since there was a successful guess on the password).

Note that the value of  $K$  is computed through an explicit oracle query. It is clear that the value of  $K$  produced in this case are the same as would be computed in the real system.

### 3.7.2 Proofs of Claims

*Proof of Claim 1.* We will call an oracle query of form  $H_s(A, B, m, \mu, \sigma, \pi)$ , for  $s \in \{2a, 2b\}$ , “bad” if  $\sigma = \text{DH}(\mu, \frac{m}{(H_1(A, B, \pi))^r})$ .

Suppose that with some nonnegligible probability  $\epsilon$  there will be some responder instance  $(\hat{i}', \hat{j}')$  (with  $B = ID_{\hat{i}'}$  and  $A = PID_{\hat{i}', \hat{j}'}$ ) such that the following “bad event” occurs:

1. query  $B1(\hat{m})$  is made to  $(\hat{i}', \hat{j}')$  and returns  $\hat{\mu}$ , and
2. at least two “bad” queries are made with  $(A, B, \hat{m}, \hat{\mu})$  and distinct values of  $\pi$ , before there is a successful guess on  $\{A, B\}$ .

We will then show how to construct a distinguisher  $D$  for the DDH problem.

The idea of the distinguisher is as follows: We start with a triple  $(X, Y, Z)$  as input, for which we need to determine whether or not  $Z = \text{DH}(X, Y)$ . We will run the adversary against a *simulation* of the original PAK simulator. In the simulation, we will use  $X$  and  $Y$  in place of some random values, and, if the adversary can cause a “bad event,” we will get information about  $\text{DH}(X, Y)$ . More specifically, we will

“incorporate”  $X$  into half of the  $H_1$  responses (randomly), and  $Y$  into  $B_1$  responses (i.e., the  $\mu$  values). A bad query will then have to contain

$$\sigma = \text{DH}(\mu, \frac{m}{(H_1(A, B, \pi))^r}) = \text{DH}(Yg^z, \frac{m}{X^{b[\pi]r}g^\alpha}),$$

where  $b[\pi] \in \{0, 1\}$ ,  $z$  and  $\alpha$  are random values known to us, and  $m$  comes from the adversary. One bad query does not give us information about  $\text{DH}(X, Y)$ , since we don’t know  $m$ . However, two bad queries with different values of  $b[\pi]$  will give us enough information. If  $b[\pi]$  is randomly chosen for each  $\pi$ , and the adversary asks two bad queries for different choices of the password, then we will get information about  $\text{DH}(X, Y)$ . The main difficulty in the construction is how to perform the simulation without knowing the discrete logarithms of  $H_1$  and  $\mu$  values. For that reason, we want to minimize the number of places where  $X$  and  $Y$  get used, and so we need to make a guess as to where the “bad event” will occur.

Now let us give the details. Our distinguisher  $D$  for input  $(X, Y, Z)$  runs as follows:

1. Generate random  $d$  between 1 and  $T$ .
2. Initialize two lists  $\text{BAD}_0$  and  $\text{BAD}_1$  (initially empty).
3. We will be running the simulator in the normal manner, but playing the ring master also. (That is, we will choose passwords for user pairs, and answer any test password queries.) Run the simulation until the  $d$ th pair of users  $(A, B)$  is mentioned. (This may be from an oracle query  $H_1(A, B, \cdot)$ , or from an *initialize user instance* query with user ID  $A$  and partner ID  $B$ , or vice-versa.) If we guessed  $d$  correctly, this pair will be the identities of the users in the “bad event.”
4. Once  $A$  and  $B$  are set, continue as in the original simulator, except:
  - (a)  $\text{B1}(m)$  query to instance  $(i', j')$  with  $ID_{i'} = B$  and  $PID_{i'j'} = A$ : generate  $z_{i'j'} \xleftarrow{R} Z_q$  and  $k_{i'j'} \xleftarrow{R} \{0, 1\}^\kappa$ , set  $\mu = Yg^{z_{i'j'}}$ , and respond with  $(\mu, k_{i'j'})$ .

- (b)  $H_1(A, B, \pi)$ : If  $\pi = \pi^*$ , then set  $b[\pi] = b[\pi^*] = 0$ . Otherwise, let  $b[\pi] \xleftarrow{R} \{0, 1\}$ . Respond with  $(X^{b[\pi]} \cdot h^q g^{\alpha[A, B, \pi]} \bmod p) + \beta p$ , for  $\alpha[A, B, \pi] \xleftarrow{R} Z_q$ ,  $h \xleftarrow{R} Z_p^*$ , and  $\beta \xleftarrow{R} Z_{\lfloor 2^n/p \rfloor}$ .
- (c)  $A2(\mu, k)$  query to initiator instance  $(i, j)$ , where  $ID_i = A$  and  $PID_{ij} = B$ : Behave as in the original simulator, except if we get into case 2, then look for query

$$H_{2a}(A, B, m, \mu, \mu^{w-r\alpha[A, B, \pi^*]}, \pi^*),$$

i.e., ignore any  $\pi \neq \pi^*$ .

Note that we know  $\pi^*$ , and that  $(H_1(A, B, \pi^*))^r = g^{r \cdot \alpha[A, B, \pi^]}$ , since  $b[\pi^*] = 0$ . Thus, the original simulator will work correctly in this case. It is safe to ignore oracle queries with  $\pi \neq \pi^*$ , since those wouldn't lead to an accept (as the *test instance password* query would fail).

- (d)  $B3(k')$  query to responder instance  $(i', j')$ , where  $PID_{i'j'} = A$  and  $ID_{i'} = B$ : If there is no matching conversation with an acceptable partner, then respond with reject. Otherwise, behave as in the original simulator.

If the “bad event” is about to occur and  $d$  has been guessed correctly, then this response is appropriate: If there was no matching conversation with an acceptable partner and this query was supposed to result in an accept, then there would be a successful guess on  $\{A, B\}$  before the second “bad” oracle query, and that would contradict the definition of the “bad event.” On the other hand, if the “bad event” has already occurred, then we don't care whether or not the response was correct (as will be seen below, it is sufficient for us to have the “bad event” at any point in our execution, and we don't need to know the point at which it actually occurred).

- (e)  $(A, B, m, \mu, \sigma, \pi)$  query to  $H_{2a}$  or  $H_{2b}$ :

If  $H_1(A, B, \pi)$  was queried and there is an instance  $(i', j')$  with  $B = ID_{i'}$  that was queried with  $B1(m)$  and returned  $\mu = Yg^{z_{i'j'}}$ , then do the following (and otherwise, answer the query as in the original simulator):

First note that if this is a bad query, then

$$\sigma = DH(\mu, \frac{m}{(H_1(A, B, \pi))^r}) = DH(Yg^{z_{i'j'}}, \frac{m}{X^{b[\pi]r}g^{r \cdot \alpha[A, B, \pi]}}).$$

Now compute

$$\gamma = \sigma m^{-z_{i'j'}} X^{b[\pi]r z_{i'j'}} Z^{b[\pi]r} Y^{r \alpha[A, B, \pi]} g^{r \cdot \alpha[A, B, \pi] z_{i'j'}}.$$

Put  $\gamma$  on the list  $BAD_{b[\pi]}$ . Then respond with a random  $k$ .

If the bad event is about to occur, then this response will be indistinguishable from the original simulator, since no successful guess on  $\{A, B\}$  can occur before the bad event.

At the end of the simulation, check whether the lists  $BAD_0$  and  $BAD_1$  intersect (note that this check can be done in time  $O(T \log T)$ , by sorting the lists together). If yes, then output “True DH,” otherwise output “Random.”

Note that the values stored in  $BAD_0$  and  $BAD_1$  are simply guesses of  $DH(m, Y)$ , assuming  $Z = DH(X, Y)$ . If  $Z$  is random, then the probability of an intersection between the lists would be at most  $T^2/q$  by the union bound, since  $Z^r$  would be a random element of  $G_{p,q}$  (note that  $q$  and  $r$  are relatively prime, and  $m \neq 0 \pmod p$  because of the test by  $B$ ).

On the other hand, suppose  $Z = DH(X, Y)$ . If the adversary makes two “bad” queries for the pair of users  $(A, B)$ , for passwords  $\pi_1, \pi_2$  with  $b[\pi_1] \neq b[\pi_2]$ , then the distinguisher will correctly answer “True DH” (since each “bad” query will result in  $\gamma = DH(m, Y)$ ). The probability of the “bad event” is  $\epsilon$ . The probability of guessing  $d$  correctly is  $\frac{1}{T}$ . The probability of  $b[\pi_1] \neq b[\pi_2]$  for  $\pi_1 \neq \pi_2$  is  $\frac{1}{2}$ . All of these events are independent.

Now, the probability that the distinguisher  $D$  guesses correctly is at least

$$\begin{aligned}
\Pr(D \text{ is correct}) &= \Pr(D \text{ guesses "DH True"} | \text{DH instance}) \Pr(\text{DH instance}) \\
&\quad + \Pr(D \text{ guesses "Random"} | \text{Random instance}) \Pr(\text{Random instance}) \\
&\geq \left(\frac{\epsilon}{2T}\right) \left(\frac{1}{2}\right) + \left(1 - \frac{T^2}{q}\right) \left(\frac{1}{2}\right).
\end{aligned}$$

Thus the probability that  $D$  is correct is at least  $\frac{1}{2} + \frac{\epsilon}{4T} - \frac{T^2}{2q}$ , which is non-negligibly more than  $\frac{1}{2}$ .  $\square$

*Proof of Claim 3.* If such a query is indeed made with some non-negligible probability  $\epsilon$ , then we will construct a distinguisher  $D$  for DDH. Let  $(X, Y, Z)$  be the challenge DDH instance.

The idea of the construction is similar to the one in the previous proof. The main difference is that now we incorporate  $X$  and  $Y$  into the  $m$  and  $\mu$  values, respectively, sent in a matching conversation.

The distinguisher runs as follows:

1. Generate random  $d$  between 1 and  $T$ , and random  $b \in \{0, 1\}$ .
2. We will be running the simulator in the normal manner, but playing the ring master also. (That is, we will choose passwords for user pairs, and answer any test password queries.) Run the simulator until the  $d$ th A0 query. Say this query is to  $(i, j)$ . Let  $A = ID_i$ ,  $B = PID_{ij}$ , and  $\pi^* = \pi[A, B]$ . Reply to this A0 query with  $m = X$ .
3. Continue with the simulation, but with the following changes:
  - (a) B1( $m$ ) query to instance  $(i', j')$  where  $m = X$ ,  $ID_{i'} = B$ , and  $PID_{i'j'} = A$ : Generate a random  $z_{i'j'} \in \mathbb{Z}_q$ , and set  $\mu = Yg^{z_{i'j'}}$ . Compute  $k = k_{i'j'}$  as in the original simulator and return  $(\mu, k)$ .
  - (b) Any  $H_{2a}$ ,  $H_{2b}$  or  $H_3$  query  $(A, B, X, \mu, \sigma, \pi)$  where  $\mu = Yg^{z_{i'j'}}$  for some  $(i', j')$  and  $\sigma = ZX^{z_{i'j'}}/\mu^{r\alpha[A, B, \pi]}$ : Stop the distinguisher and guess "True DH."

- (c) A2( $\mu, k$ ) query to  $(i, j)$  not part of a matching conversation with an acceptable partner:

Check that an  $H_1(A, B, \pi^*)$  query has been asked and a query  $H_{2a}(A, B, m, \mu, \sigma, \pi^*)$  has been asked and returned  $k$  (w.o.p., there will be at most one such  $H_{2a}$  query, by the randomness of  $H_{2a}$  responses). If not, then reject.

Otherwise, if  $b = 0$ , reject, and if  $b = 1$ ,

- i. Set  $k'_{ij} = H_{2b}(A, B, m, \mu, \sigma, \pi^*)$ .
- ii. Send  $k' = k'_{ij}$ .
- iii. Accept.
- iv. Set  $K_{ij} = H_3(A, B, m, \mu, \sigma, \pi^*)$ .
- v. Expose using  $K = K_{ij}$ .

- (d) B3( $k'$ ) query to instance  $(i', j')$  where  $m = X$ , which is not part of a matching conversation:

Reject. If the simulator should have accepted, the adversary w.o.p. would have already queried  $H_{2b}$  with the correct Diffie-Hellman value.

4. If the simulation ends without outputting “True DH,” then output “Random.”

Note that if the adversary does make a bad query, we have probability at least  $1/T$  of guessing the correct initiator user instance, and probability at least  $1/2$  of answering the A2 query to that user instance correctly (thus allowing the DDH distinguisher to continue in a way that is indistinguishable from the regular simulator).

Now, the probability that the distinguisher  $D$  guesses correctly is at least

$$\begin{aligned}
\Pr(D \text{ is correct}) &= \Pr(D \text{ guesses “DH True”} | \text{DH instance}) \Pr(\text{DH instance}) \\
&\quad + \Pr(D \text{ guesses “Random”} | \text{Random instance}) \Pr(\text{Random instance}) \\
&\geq \left(\frac{\epsilon}{2T}\right) \left(\frac{1}{2}\right) + \left(1 - \frac{T}{q}\right) \left(\frac{1}{2}\right),
\end{aligned}$$

where the term  $T/q$  comes from the probability of the adversary “guessing” a random  $Z$  correctly on a random oracle query. Thus, the probability that  $D$  guesses correctly is at least  $\frac{1}{2} + \frac{\epsilon}{4T} - \frac{T}{2q}$ , which is non-negligibly more than  $\frac{1}{2}$ .  $\square$

**Remark on the use of Computational Diffie-Hellman:** It is easy to see that the proofs could be modified to base security on the Computational Diffie-Hellman problem (i.e., the hardness of computing  $Z = \text{DH}(X, Y)$  for random  $X$  and  $Y$ ). Basically, instead of checking whether the random oracle queries match a given form, we would just guess which of the oracle queries are “right,” and thereby extract  $Z$ . However, this approach would only give  $O(\frac{\epsilon}{T^3})$  success probability in Claim 1, since we would need to guess the location of both of the bad oracle queries.

## 3.8 Security of the PPK Protocol

### 3.8.1 The Simulator

The proof of simulatability of PPK is similar in structure to that of PAK. We name the actions of user instances on *deliver message* operations as follows:

**A0** Initiator instance action to start the protocol (i.e., an A0 action for an initiator instance is generating and sending  $m$ ).

**B1** Responder instance action upon receiving the first message (i.e.,  $m$ ).

**A2** Initiator instance action upon receiving the message from the responder (i.e.,  $\mu$ ).

We will let  $(i, j)$ ,  $(i', j')$ ,  $A$ ,  $B$ , and  $\pi^*$  have the same meaning as in the proof of PAK (see the beginning of Section 3.7.1). The notions of *acceptable partner* and *matching conversation* will also retain their meaning.

The simulator will follow the same general rules as described in Section 3.7.1. Responses to *deliver message* and *random oracle* operations that do not fall under those rules are done as follows:

1.  $H_1(A, B, \pi)$

Same as for PAK: Generate  $\alpha[A, B, \pi] \xleftarrow{R} Z_q$  and store it. Then generate  $h \xleftarrow{R} Z_p^*$  and  $\beta \xleftarrow{R} Z_{[2^\eta/p]}$ , and return  $(h^q g^{\alpha[A, B, \pi]} \bmod p) + \beta p$ . Note that this will be

indistinguishable from a random bit string of length  $\eta$ , since  $h^q g^{\alpha[A,B,\pi]} \bmod p$  is a random element from  $Z_p^{*9}$  and  $\frac{2^{\eta \bmod p}}{2^\eta}$  is negligible.

2.  $H'_1(A, B, \pi)$

Same as for PAK: Generate  $\alpha'[A, B, \pi] \xleftarrow{R} Z_q$  and store it. Then generate  $h \xleftarrow{R} Z_p^*$  and  $\beta' \xleftarrow{R} Z_{[2^\eta/p]}$ , and return  $(h^q g^{\alpha'[A,B,\pi]} \bmod p) + \beta'p$ . Note that this will be indistinguishable from a random bit string of length  $\eta$ , since  $h^q g^{\alpha'[A,B,\pi]} \bmod p$  is a random element from  $Z_p^{*10}$  and  $\frac{2^{\eta \bmod p}}{2^\eta}$  is negligible.

3.  $H_3(A, B, m, \mu, \sigma, \pi)$

**Case 1** Some prior B1 query has recorded a tuple of the form  $(i', j', m, \mu, z)$ , with  $A = PID_{i'j'}$  and  $B = ID_{i'}$ , for some value of  $z$ . This implies that a B1( $m$ ) query was made to  $(i', j')$  and returned  $\mu$ , but no A0 query to an acceptable partner returned  $m$ —see the B1 query description below. (In other words, this  $H_3$  query might be shadowed by the implicit  $H_3$  query from case 3 of the B1 action.) Instance  $(i', j')$  must have already opened a session with the *dangling open* connection assignment (see case 3 of B1). W.o.p., there will be at most one such tuple, since  $\mu$  values stored in these tuples are random and independent. If queries  $H_1(A, B, \pi)$  and  $H'_1(A, B, \pi)$  have been asked, and  $(\frac{m}{(H_1(A,B,\pi))^r})^{z-r\alpha'[A,B,\pi]} = \sigma$ , then:

- (a) If a *test instance password* operation has not previously been performed on  $(i', j')$ , perform a *test instance password* operation with arguments  $(i', j')$  and  $\pi$ . If the test is successful, the ring master will give us the key  $K_{i'j'}$  of instance  $(i', j')$  (since  $(i', j')$  has a dangling connection assignment). Return  $K_{i'j'}$ .
- (b) If a *test instance password* operation with  $\pi$  has already been performed on  $(i', j')$  and was successful, then let  $K_{i'j'}$  be the key obtained

---

<sup>9</sup>To see this, note that  $h^q$  is a random element from the subgroup of order  $r$  in  $Z_p^*$  and  $g^{\alpha[A,B,\pi]}$  is a random element of the subgroup of order  $q$  in  $Z_p^*$ .

<sup>10</sup>To see this, note that  $h^q$  is a random element from the subgroup of order  $r$  in  $Z_p^*$  and  $g^{\alpha'[A,B,\pi]}$  is a random element of the subgroup of order  $q$  in  $Z_p^*$ .

from the ring master in that operation. Return  $K_{i'j'}$ .

- (c) If an unsuccessful *test instance password* operation has already been performed on  $(i', j')$  with a password  $\pi' \neq \pi$ , then abort. We will call this event a *responder failure* (note that  $(i', j')$  is a responder instance).

If we have neither failed nor returned anything yet, then return  $k \xleftarrow{R} \{0, 1\}^\kappa$ .

If no responder failure occurs as a result of this  $H_3$  query, then the simulation will be indistinguishable from the real world, as follows: In the real world the key generated by  $(i', j')$  is

$$H_3(A, B, m, \mu, \text{DH}(\frac{\mu}{(H'_1(A, B, \pi^*))^r}, \frac{m}{(H_1(A, B, \pi^*))^r}), \pi^*).$$

Therefore, if this  $H_3$  query is successful, it will return a value consistent with the adversary's view. On the other hand, if this query is not successful, then generating a random response is indistinguishable from the adversary's view in the real world.

The following claim shows that w.o.p.  $\mathcal{A}$  will not cause a responder failure.

**Claim 4.** *W.o.p., no responder failure will occur.*

The proof appears in Section 3.8.2. Note that case 1 of  $H_3$  is the only place in the simulation where a responder failure can occur.

As can be seen, an  $H_3$  query is the only place in the simulation that a *test instance password* query can be made on a responder instance. We can therefore state the following claim (similarly to PAK):

**Claim 5.** *If a test instance password query is made on a responder instance  $(i', j')$ , then the  $m$  value received by  $(i', j')$  was not sent by an acceptable partner.*

The reason is that an  $H_3$  query can result in a *test instance password* query on  $(i', j')$  only if a tuple  $(i', j', m, \mu, z)$  has previously been recorded. It will become apparent below (see the B1 query description) that, w.o.p., this

could only happen if the  $m$  value received by  $(i', j')$  was not sent by an acceptable partner.

**Case 2** Some prior A2 query has recorded a tuple of the form  $(i, j, m, \mu, w)$ , with  $A = ID_i$  and  $B = PID_{ij}$ , for some value of  $w$ . This implies that  $(i, j)$  sent  $m$  and received  $\mu$  back, but did not have a matching conversation with an acceptable partner—see the A2 query description below. (In other words, this  $H_3$  query might be shadowed by the implicit  $H_3$  query from case 2a or 2b of the A2 action.) Instance  $(i, j)$  must have already opened a session with the *dangling connect* connection assignment (see cases 2a and 2b of A2).

(Note that this case cannot overlap with the previous one, i.e., w.o.p., we won't have both a tuple  $(i', j', m, \mu, z)$  and tuple  $(i, j, m, \mu, w)$  recorded, with matching values of  $A$  and  $B$ . That reason is that if the values of  $m$  and  $\mu$  in the tuples match up, then  $(i, j)$  and  $(i', j')$  will have had a matching conversation. Also, the two instances are acceptable partners. However, an initiator instance that has had a matching conversation with an acceptable partner will not record a tuple—it will be in case 1 of A2, while an initiator tuple can only be recorded in case 2 of A2.)

W.o.p., there will be at most one such tuple, since  $m$  values stored in these tuples are random and independent. If queries  $H_1(A, B, \pi)$  and  $H'_1(A, B, \pi)$  have been asked, and  $(\frac{\mu}{(H'_1(A, B, \pi))^r})^{w - r\alpha[A, B, \pi]} = \sigma$ , then:

- (a) If a *test instance password* operation has not previously been performed on  $(i, j)$ , perform a *test instance password* operation with arguments  $(i, j)$  and  $\pi$ . If the test is successful, the ring master will give us the key  $K_{ij}$  of instance  $(i, j)$  (since  $(i, j)$  has a dangling connection assignment). Return  $K_{ij}$ .
- (b) If a *test instance password* operation with  $\pi$  has already been performed on  $(i', j')$  and was successful, then let  $K_{i'j'}$  be the key obtained from the ring master in that operation. Return  $K_{i'j'}$ .

- (c) If an unsuccessful *test instance password* operation has already been performed on  $(i, j)$ , then abort. We will call this event an *initiator failure* (note that  $(i, j)$  is an initiator instance; in addition, see case 2c of A2 for another place where an initiator failure can occur).

If we have neither failed nor returned anything yet, then return  $k \xleftarrow{R} \{0, 1\}^\kappa$ .

If no initiator failure occurs as a result of this  $H_3$  query, then the simulation will be indistinguishable from the real world, similarly to case 1.

The following claim shows that w.o.p.  $\mathcal{A}$  will not cause an initiator failure.

**Claim 6.** *W.o.p., no initiator failure will occur (whether in case 2 of  $H_3$  or in case 2c of A2).*

The proof appears in Section 3.8.2.

Note that the *test instance password* query on  $(i, j)$  will only be made here if a tuple  $(i, j, m, \mu, w)$  has previously been recorded. That, in turn, implies w.o.p. that  $(i, j)$  did not have a matching conversation with an acceptable partner (see the note at the end of case 2b of A2). We have just proven one part of the following claim:

**Claim 7.** *If a test instance password query is made on an originator instance  $(i, j)$ , then  $(i, j)$  has not had a matching conversation with an acceptable partner.*

The other part is shown in case 2a of A2 (which is the only other place where a *test instance password* could be made on an originator instance).

**Case 3** Otherwise.

Return  $k \xleftarrow{R} \{0, 1\}^\kappa$ .

Let's show that this response is valid, i.e., that this  $H_3$  query can't be shadowed by any prior implicit queries. Implicit  $H_3$  queries are made in cases 1, 2, and 3 of B1, and cases 2a and 2b of A2. It is easy to see that any  $H_3$  call that might shadow an implicit query from case 3 of B1 would

not be dealt with here (rather, it would be dealt with in case 1 of  $H_3$ ). Similarly, we don't need to be concerned for implicit queries from cases 2a and 2b of A2, since they are only relevant to case 2 of  $H_3$ . Thus, we only need to show that there is no shadowing from cases 1 and 2 of B1. These are dealt with by the following:

**Claim 8.** *Let  $m$  be returned by an A0 query to  $(i, j)$  with  $A = ID_i$  and  $B = PID_{ij}$ , and  $\mu$  be returned by a subsequent B1( $m$ ) query to  $(i', j')$  with  $B = ID_{i'}$  and  $A = PID_{i'j'}$ . Then, w.o.p., there will never be a query  $H_3(A, B, m, \mu, \text{DH}(\frac{\mu}{(H'_1(A, B, \pi))^r}, \frac{m}{(H_1(A, B, \pi))^r}), \pi)$ , for any  $\pi$ .*

The proof of the claim appears in Section 3.8.2.

4. A0 query to  $(i, j)$

Same as for PAK: Generate and store  $w \xleftarrow{R} Z_q$ , and send  $m = g^w$ . Clearly,  $m$  is uniformly drawn from  $G_{p,q}$ , just as in the real system.

5. B1( $m$ ) query to  $(i', j')$

Generate and store  $z \xleftarrow{R} Z_q$ , and send  $\mu = g^z$  (which clearly has the correct distribution). Now consider three cases:

**Case 1** The value of  $m$  has been sent by an acceptable partner  $(i, j)$ , and no A2 action has yet been performed for  $(i, j)$ . (Note that by the randomness of  $m$  values, there will, w.o.p., be at most one such  $(i, j)$ .)

Set status to Accept, and perform a *start session* operation with the connection assignment *open for connection from  $(i, j)$* . This corresponds to the implicit oracle query

$$K_{i'j'} = H_3 \left( A, B, m, \mu, \text{DH} \left( \frac{m}{(H_1(A, B, \pi^*))^r}, \frac{\mu}{(H'_1(A, B, \pi^*))^r} \right), \pi^* \right),$$

where  $K_{i'j'}$  is the random session key assigned by the ring master at the time of the *start session* operation. Since  $\mu$  is a freshly chosen random

number, this implicit query, w.o.p., will not shadow any prior  $H_3$  queries (whether explicit or implicit). Consequently,  $K_{i'j'}$  will be indistinguishable from the value used in the real system.

The connection assignment is legal because, by Claim 5, there could not have been a *test instance password* query on  $(i', j')$ , since  $(i', j')$  has received  $m$  from an acceptable partner.

**Case 2** The value of  $m$  has been sent by an acceptable partner  $(i, j)$ , and an  $A2(\mu')$  action has been performed for  $(i, j)$ , for some  $\mu'$ .

Set status to Accept, and perform a *start session* operation with the connection assignment *dangling open*. This corresponds to the implicit oracle query

$$K_{i'j'} = H_3 \left( A, B, m, \mu, \text{DH} \left( \frac{m}{(H_1(A, B, \pi^*))^r}, \frac{\mu}{(H'_1(A, B, \pi^*))^r} \right), \pi^* \right),$$

where  $K_{i'j'}$  is the random session key assigned by the ring master at the time of the *start session* operation. As in case 1, it is easy to see that the values of  $\mu$  and  $K_{i'j'}$  will be indistinguishable from those used in the real system.

**Case 3** The value of  $m$  has not been sent by an acceptable partner.

Record the tuple  $(i', j', m, \mu, z)$ . Then set status to Accept, and perform a *start session* operation with the connection assignment *dangling open*. This corresponds to the implicit oracle query

$$K_{i'j'} = H_3 \left( A, B, m, \mu, \text{DH} \left( \frac{m}{(H_1(A, B, \pi^*))^r}, \frac{\mu}{(H'_1(A, B, \pi^*))^r} \right), \pi^* \right),$$

and is valid for the same reason as in case 2.

It is important to note (for Claim 5), that if a tuple is recorded, then, w.o.p., the  $m$  value received by  $(i', j')$  neither was nor every will be sent

by an acceptable partner, due to the fact that values of  $m$  are generated randomly by initiator instances.

6.  $A2(\mu)$  query to  $(i, j)$

Consider two cases:

**Case 1**  $(i, j)$  has had a matching conversation with an acceptable partner  $(i', j')$ .

Set status to Accept and perform a *start session* operation with a *connect* to  $(i', j')$  connection assignment.

Let's show that this connection assignment is legal. Since  $(i, j)$  had a matching conversation with  $(i', j')$ , it follows that  $(i', j')$  has received the  $m$  sent by  $(i, j)$ , and  $(i, j)$  received the  $\mu$  sent by  $(i', j')$ . Since  $\mu$  was a fresh random number generated by  $(i', j')$ , it follows that w.o.p.  $(i, j)$  received  $\mu$  after  $(i', j')$  sent it. It is now easy to see that the  $B2(m)$  action for  $(i', j')$  was, w.o.p., in case 1. Thus,  $(i', j')$  was open for connection from  $(i, j)$ . W.o.p.,  $(i', j')$  became open for connection from  $(i, j)$  after  $(i, j)$  sent  $m$ , since  $m$  was a fresh random number.

In addition, by Claim 7, no *test instance password* could have been performed on  $(i, j)$ , since  $(i, j)$  has had a matching conversation with an acceptable partner.

**Case 2**  $(i, j)$  has not had a matching conversation with an acceptable partner.

Now consider two cases:

**Case 2a** There exists exactly one  $\pi$  such that queries  $H_1(A, B, \pi)$  and

$H'_1(A, B, \pi)$  have been asked, and query  $H_3(A, B, m, \mu, (\frac{\mu}{(H'_1(A, B, \pi))^r})^{w-r\alpha[A, B, \pi]}, \pi)$  has been asked and returned  $K$ .

Perform a *test instance password* operation with arguments  $(i, j)$  and  $\pi$ . (Note that in this case  $(i, j)$  has not had a matching conversation with an acceptable partner. This completes the proof of Claim 7.)

This is legal for the following reason: The only other place where a

*test instance password* could be asked on an originator instance is in case 2 of  $H_3$ . However, that case can only lead to a test if a tuple  $(i, j, \dots)$  has been recorded. Such a tuple can only be recorded by an A2 action. Now, clearly no tuple has been recorded thus far by this A2 action, and also no A2 action could have occurred on  $(i, j)$  before (we don't allow more than one A2 action to be performed on any particular instance).

- (a) If the test is successful, then set status to Accept and expose using key  $K$ . This is acceptable for the same reasons as above.
- (b) Otherwise, record the tuple  $(i, j, m, \mu, w)$ , set status to Accept, and perform a *start session* operation with the connection assignment *dangling connect*.

This corresponds to the implicit oracle query

$$K_{ij} = H_3 \left( A, B, m, \mu, \text{DH} \left( \frac{m}{(H_1(A, B, \pi^*))^r}, \frac{\mu}{(H'_1(A, B, \pi^*))^r} \right), \pi^* \right),$$

where  $K_{ij}$  is the random session key assigned by the ring master at the time of the *start session* operation. Clearly, this implicit query will not, w.o.p., shadow any prior explicit queries, or else we would have exposed. W.o.p., it won't shadow another implicit query from an A2 action, by the randomness of  $m$  values. Also, it won't shadow an implicit query from a B1 action of any instance  $(i', j')$ , since that would imply that  $(i, j)$  had a matching conversation with an acceptable partner (as  $A$ ,  $B$ ,  $m$ , and  $\mu$  would have to match), and so we would not be in case 2 of A2.

**Case 2b** No such  $\pi$  exists.

Record the tuple  $(i, j, m, \mu, w)$ . Then set status to Accept, and perform a *start session* operation with the connection assignment *dangling*

*connect*. This corresponds to the implicit oracle query

$$K_{ij} = H_3 \left( A, B, m, \mu, \text{DH} \left( \frac{m}{(H_1(A, B, \pi^*))^r}, \frac{\mu}{(H'_1(A, B, \pi^*))^r} \right), \pi^* \right),$$

and is valid for the same reason as above at the end of case 2a.

It is important to note (for Claim 7), that if a tuple is recorded (either in case 2a or in case 2b), then, w.o.p.,  $(i, j)$  will never be considered to have a matching conversation with an acceptable partner, due to the fact that values of  $\mu$  are generated randomly by responder instances. (Recall that in our definition of “matching conversation” we are not concerned with the timing of sends and receives.)

**Case 2c** There is more than one such  $\pi$ .

Abort with an *initiator failure*. W.o.p., this case will not occur, by Claim 6.

### 3.8.2 Proofs of Claims

*Proof of Claim 4.* We will call an oracle query of form  $H_3(A, B, m, \mu, \sigma, \pi)$  “bad” if  $\sigma = \text{DH}(\frac{\mu}{(H'_1(A, B, \pi))^r}, \frac{m}{(H_1(A, B, \pi))^r})$ .

Suppose that with some nonnegligible probability  $\epsilon$  there will be a responder failure. This implies that there will be some responder instance  $(\hat{i}', \hat{j}')$  (with  $B = ID_{\hat{i}'}$  and  $A = PID_{\hat{i}'\hat{j}'}$ ) such that the following “bad event” occurs:

1. query  $\text{B1}(\hat{m})$  is made to  $(\hat{i}', \hat{j}')$  and returns  $\hat{\mu}$ , and
2. at least two “bad” queries are made with  $(A, B, \hat{m}, \hat{\mu})$  and distinct values of  $\pi$ , before there is either a successful *test instance password* on  $(\hat{i}', \hat{j}')$  or an initiator failure. (Note that the simulator aborts when it encounters an initiator failure. Thus, if a responder failure occurs, that implies that no initiator failure has occurred so far.)

We will then show how to construct a distinguisher  $D$  for the DDH problem.

The idea of our distinguisher  $D$ , with input  $(X, Y, Z)$ , is to simulate the original simulator, incorporating  $X$  and  $Y$  into some of the responses. Then, if  $D$  runs indistinguishably from the simulator up to the bad event, we will be able to use the logs to determine whether or not  $Z = \text{DH}(X, Y)$ .

Our distinguisher  $D$  for input  $(X, Y, Z)$  runs as follows:

1. Generate random  $d_1$  and  $d_2$  between 1 and  $T$ .
2. Initialize two lists  $\text{BAD}_0$  and  $\text{BAD}_1$  (initially empty).
3. We will be running the simulator in the normal manner, but playing the ring master also. (That is, we will choose passwords for user pairs, and answer any test password queries.) Run the simulation until the  $d_1$ th pair of users  $(A, B)$  is mentioned. (This may be from an oracle query  $H_1(A, B, \cdot)$ , or from an *initialize user instance* query with user ID  $A$  and partner ID  $B$ , or vice-versa.) If we guessed  $d_1$  correctly, this pair will be the identities of the users in the “bad event.”
4. Once  $A$  and  $B$  are set, continue as in the original simulator, except:
  - (a)  $\text{B1}(m)$  query to the  $d_2$ th responder instance  $(\hat{i}', \hat{j}')$  with  $\text{ID}_{\hat{i}'} = B$  and  $\text{PID}_{\hat{i}'\hat{j}'} = A$ : respond with  $\hat{\mu} = Y$ . Let  $\hat{m} = m$ .  
If we guessed  $d_2$  correctly, this will be the responder instance in the “bad event.” Note that it is OK to set  $\hat{\mu} = Y$ , since  $Y$  is assumed to be uniformly distributed.
  - (b)  $H_1(A, B, \pi)$ : If  $\pi = \pi^*$ , then set  $b[\pi] = b[\pi^*] = 0$ . Otherwise, let  $b[\pi] \xleftarrow{R} \{0, 1\}$ . Respond with  $(X^{b[\pi]} \cdot h^q g^{\alpha[A, B, \pi]} \bmod p) + \beta p$ , for  $h \xleftarrow{R} Z_p^*$ ,  $\alpha[A, B, \pi] \xleftarrow{R} Z_{p-1}$  and  $\beta \xleftarrow{R} Z_{\lfloor 2^n/p \rfloor}$ .
  - (c)  $\text{A2}(\mu)$  query to initiator instance  $(i, j)$ , where  $\text{ID}_i = A$  and  $\text{PID}_{ij} = B$ : Behave as in the original simulator, except if we get into case 2, then look for query

$$H_3(A, B, m, \mu, (\frac{\mu}{(H'_1(A, B, \pi^*))^r})^{w-r\alpha[A, B, \pi^*]}, \pi^*),$$

i.e., ignore any  $\pi \neq \pi^*$ .

Note that we know  $\pi^*$ , and that  $(H_1(A, B, \pi^*))^r = g^{r \cdot \alpha[A, B, \pi^*]}$ , since  $b[\pi^*] = 0$ . Thus, the original simulator will work correctly in this case. It is safe to ignore oracle queries with  $\pi \neq \pi^*$ , since those wouldn't lead to an accept (as the *test instance password* query would fail).

It is easy to see that the only way this behavior might be distinguishable from the original simulator is if this query was supposed to cause an abort (i.e., an initiator failure). However, this is not a problem: If the bad event has already occurred, then we are no longer interested in preserving indistinguishability. On the other hand, if the bad event is still about to occur, then this query couldn't cause an initiator failure (since the bad event, by definition, can't occur after an initiator failure).

- (d)  $H_3(A, B, m, \mu, \sigma, \pi)$  query, with a tuple  $(i, j, m, \mu, w)$  recorded,  $A = ID_i$ ,  $B = PID_{ij}$ : If  $\pi = \pi^*$ , then behave as in the original simulator (this will be correct, since  $b[\pi^*] = 0$ ). Otherwise, return  $k \xleftarrow{R} \{0, 1\}^\kappa$ .

This behavior is indistinguishable from the original simulator for the same reasons as mentioned in the A2 query.

- (e)  $H_3(A, B, \hat{m}, \hat{\mu}, \sigma, \pi)$  query (note that this case will not, w.o.p., overlap with the preceding one for reasons similar to those mentioned in case 2 of  $H_3$  in the description of the original simulator, i.e., that it would imply a matching conversation with an acceptable partner, and thus the tuple  $(i, j, \dots)$  wouldn't have been recorded):

If  $H_1(A, B, \pi)$  and  $H'_1(A, B, \pi)$  were queried, then do the following (and otherwise, answer the query as in the original simulator):

First note that if this is a bad query, then

$$\begin{aligned} \sigma &= DH \left( \frac{\hat{m}}{(H_1(A, B, \pi))^r}, \frac{\hat{\mu}}{(H'_1(A, B, \pi))^r} \right) \\ &= DH \left( \frac{\hat{m}}{X^{b[\pi]r} g^{r \cdot \alpha[A, B, \pi]}}, \frac{Y}{g^{r \cdot \alpha'[A, B, \pi]}} \right). \end{aligned}$$

Now compute

$$\gamma = \sigma \cdot Z^{b[\pi]r} \cdot Y^{r\alpha[A,B,\pi]} \cdot (\hat{m} \cdot X^{-b[\pi]r} \cdot g^{-r\alpha[A,B,\pi]})^{r\alpha'[A,B,\pi]}.$$

Put  $\gamma$  on the list  $\text{BAD}_{b[\pi]}$ . Then respond with a random  $k$ .

If the bad event is about to occur, then this response will be indistinguishable from the original simulator, since no successful *test instance password* on  $(\hat{i}', \hat{j}')$  can occur before the bad event.

At the end of the simulation, check whether the lists  $\text{BAD}_0$  and  $\text{BAD}_1$  intersect (note that this check can be done in time  $O(T \log T)$ , by sorting the lists together). If yes, then output “True DH,” otherwise output “Random.”

Note that the values stored in  $\text{BAD}_0$  and  $\text{BAD}_1$  are simply guesses of  $DH(\hat{m}, Y)$ , assuming  $Z = DH(X, Y)$ . If  $Z$  is random, then the probability of an intersection between the lists would be at most  $T^2/q$  by the union bound, since  $Z^r$  would be a random element of  $G_{p,q}$  (note that  $q$  and  $r$  are relatively prime, and  $m \neq 0 \pmod p$  because of the test by  $B$ ).

On the other hand, suppose  $Z = DH(X, Y)$ . If the adversary makes two “bad” queries for the pair of users  $(A, B)$ , for passwords  $\pi_1, \pi_2$  with  $b[\pi_1] \neq b[\pi_2]$ , then the distinguisher will correctly answer “True DH” (since each “bad” query will result in  $\gamma = DH(\hat{m}, Y)$ ). The probability of the “bad event” is  $\epsilon$ . The probability of guessing  $d_1$  and  $d_2$  correctly is  $\frac{1}{T^2}$ . The probability of  $b[\pi_1] \neq b[\pi_2]$  for  $\pi_1 \neq \pi_2$  is  $\frac{1}{2}$ . All of these events are independent.

Now, the probability that the distinguisher  $D$  guesses correctly is at least

$$\begin{aligned} \Pr(D \text{ is correct}) &= \Pr(D \text{ guesses “DH True”} | \text{DH instance}) \Pr(\text{DH instance}) \\ &\quad + \Pr(D \text{ guesses “Random”} | \text{Random instance}) \Pr(\text{Random instance}) \\ &\geq \left( \frac{\epsilon}{2T^2} \right) \left( \frac{1}{2} \right) + \left( 1 - \frac{T^2}{q} \right) \left( \frac{1}{2} \right). \end{aligned}$$

Thus the probability that  $D$  is correct is at least  $\frac{1}{2} + \frac{\epsilon}{4T^2} - \frac{T^2}{2q}$ , which is non-negligibly more than  $\frac{1}{2}$ .  $\square$

*Proof of Claim 6.* We will call an oracle query of form  $H_3(A, B, m, \mu, \sigma, \pi)$  “bad” if  $\sigma = \text{DH}(\frac{\mu}{(H_1(A, B, \pi))^r}, \frac{m}{(H_1(A, B, \pi))^r})$ .

Suppose that with some nonnegligible probability  $\epsilon$  there will be an initiator failure. This implies that there will be some initiator instance  $(\hat{i}, \hat{j})$  (with  $A = ID_{\hat{i}}$  and  $B = PID_{\hat{j}}$ ) such that the following “bad event” occurs:

1. query A0 is made to  $(\hat{i}, \hat{j})$  and returns  $\hat{m}$ ,
2. at least two “bad” queries are made with  $(A, B, \hat{m}, \hat{\mu})$  and distinct values of  $\pi$ , for some  $\hat{\mu}$ , before there is either a successful *test instance password* on  $(\hat{i}, \hat{j})$  or a responder failure, and
3. at some point in the execution, the adversary asks the query  $A2(\hat{\mu})$  to  $(\hat{i}, \hat{j})$ , and it is processed in case 2 of A2 (this could be either before, in between, or after the bad queries).

We will then show how to construct a distinguisher  $D$  for the DDH problem.

Our distinguisher  $D$  for input  $(X, Y, Z)$  runs as follows:

1. Generate random  $d_1$  and  $d_2$  between 1 and  $T$ .
2. Initialize two arrays of lists  $\text{BAD}_0[\cdot]$  and  $\text{BAD}_1[\cdot]$  (initially empty).
3. We will be running the simulator in the normal manner, but playing the ring master also. (That is, we will choose passwords for user pairs, and answer any test password queries.) Run the simulation until the  $d_1$ th pair of users  $(A, B)$  is mentioned. (This may be from an oracle query  $H_1(A, B, \cdot)$ , or from an *initialize user instance* query with user ID  $A$  and partner ID  $B$ , or vice-versa.) If we guessed  $d_1$  correctly, this pair will be the identities of the users in the “bad event.”
4. Once  $A$  and  $B$  are set, continue as in the original simulator, except:
  - (a) A0 query to the  $d_2$ th initiator instance  $(\hat{i}, \hat{j})$  with  $ID_{\hat{i}} = B$  and  $PID_{\hat{j}} = A$ : respond with  $\hat{m} = Y$ .

If we guessed  $d_2$  correctly, this will be the initiator instance in the “bad event.” Note that it is OK to set  $\hat{m} = Y$ , since  $Y$  is assumed to be uniformly distributed.

- (b)  $H'_1(A, B, \pi)$ : If  $\pi = \pi^*$ , then set  $b[\pi] = b[\pi^*] = 0$ . Otherwise, let  $b[\pi] \xleftarrow{R} \{0, 1\}$ . Respond with  $(X^{b[\pi] \cdot h^q g^{\alpha'[A, B, \pi]} \bmod p} + \beta' p, \text{ for } h \xleftarrow{R} Z_p^*, \alpha'[A, B, \pi] \xleftarrow{R} Z_{p-1} \text{ and } \beta' \xleftarrow{R} Z_{[2^n/p]}.$
- (c)  $A2(\mu)$  on  $(\hat{i}, \hat{j})$ : Set  $\hat{\mu} = \mu$ . Then do *start session* with the *dangling connect* connection assignment.

Note that if the bad event is about to occur, then we don't need to be concerned for case 1 of A2. Also, the original simulator would not expose, since that would imply a successful *test instance password* on  $(\hat{i}, \hat{j})$ , and that is also ruled by the bad event. Now, it is easy to see that the only way the above behavior might be distinguishable from the original simulator is if this query was supposed to cause an abort (i.e., an initiator failure). However, this would imply that the bad event has already occurred (i.e., the two bad queries have already been asked), and so we are no longer interested in preserving indistinguishability.

- (d)  $H_3(A, B, m, \mu, \sigma, \pi)$  query, with a tuple  $(i', j', m, \mu, z)$  recorded,  $A = PID_{i'j'}$ ,  $B = ID_{i'}$ : If  $\pi = \pi^*$ , then behave as in the original simulator (this will be correct, since  $b[\pi^*] = 0$ ). Otherwise, return  $k \xleftarrow{R} \{0, 1\}^\kappa$ .

It is easy to see that the only way this behavior might be distinguishable from the original simulator is if this query was supposed to cause an abort (i.e., a responder failure). However, this is not a problem: If the bad event has already occurred, then we are no longer interested in preserving indistinguishability. On the other hand, if the bad event is still about to occur, then this query couldn't cause a responder failure (since the bad event, by definition, can't occur after a responder failure).

- (e)  $H_3(A, B, \hat{m}, \mu, \sigma, \pi)$  query (note that this case will not, w.o.p., overlap with the preceding one since it would imply that some responder instance

$(i', j')$  has received its  $m$  from an acceptable partner, and yet recorded a tuple—that is impossible, since cases 1 and 2 of B1 do not record tuples): If  $H_1(A, B, \pi)$  and  $H'_1(A, B, \pi)$  were queried, then do the following (and otherwise, answer the query as in the original simulator):

First note that if this is a bad query, then

$$\begin{aligned}\sigma &= DH\left(\frac{\hat{m}}{(H_1(A, B, \pi))^r}, \frac{\mu}{(H'_1(A, B, \pi))^r}\right) \\ &= DH\left(\frac{Y}{g^{r \cdot \alpha[A, B, \pi]}}, \frac{\mu}{X^{b[\pi]r} g^{r \cdot \alpha'[A, B, \pi]}}\right).\end{aligned}$$

Now compute

$$\gamma = \sigma \cdot Z^{b[\pi]r} \cdot Y^{r \alpha'[A, B, \pi]} \cdot (\mu \cdot X^{b[\pi]r} \cdot g^{r \cdot \alpha'[A, B, \pi]})^{r \alpha[A, B, \pi]}.$$

Put  $\beta$  on the list  $\text{BAD}_{b[\pi]}[\mu]$ . Then respond with a random  $k$ .

If the bad event is about to occur, then this response will be indistinguishable from the original simulator, since no successful *test instance password* on  $(\hat{i}, \hat{j})$  can occur before the bad event.

At the end of the simulation, check whether the lists  $\text{BAD}_0[\hat{\mu}]$  and  $\text{BAD}_1[\hat{\mu}]$  intersect (note that this check can be done in time  $O(T \log T)$ , by sorting the lists together). If yes, then output “True DH,” otherwise (and also if  $\hat{\mu}$  has never been set) output “Random.”

It is easy to show, similarly to Claim 4, that  $D$  is correct with probability at least  $\frac{1}{2} + \frac{\epsilon}{4T^2} - \frac{T^2}{2q}$ , which is non-negligibly more than  $\frac{1}{2}$ .  $\square$

*Proof of Claim 8.* If such a query is indeed made with some non-negligible probability  $\epsilon$ , then we will construct a distinguisher  $D$  for DDH. Note that, as in proofs of Claims 4 and 6, we can presume that no failures will occur before the “bad event.”

Let  $(X, Y, Z)$  be the challenge DDH instance. The distinguisher runs as follows:

1. Generate random  $d_1$  and  $d_2$  between 1 and  $T$ , and random  $b \in \{0, 1\}$ .

2. We will be running the simulator in the normal manner, but playing the ring master also. (That is, we will choose passwords for user pairs, and answer any test password queries.) Run the simulator until the  $d_1$ th A0 query. Say this query is to  $(\hat{i}, \hat{j})$ . Let  $A = ID_{\hat{i}}$ ,  $B = PID_{\hat{j}}$ , and  $\pi^* = \pi[A, B]$ . Reply to this A0 query with  $\hat{m} = X$ .

3. Continue with the simulation, but with the following changes:

(a) B1( $\hat{m}$ ) query to instance  $(i', j')$  where  $ID_{i'} = B$ , and  $PID_{j'} = A$ :

Generate a random  $z_{i'j'} \in Z_q$ , and send  $\mu = Yg^{z_{i'j'}}$ .

(b) A2( $\mu$ ) query to  $(\hat{i}, \hat{j})$  not part of a matching conversation with an acceptable partner:

The difficulty in this case is that the adversary may ask an  $H_3$  query that should give him the correct key, but we are unable to determine which of the adversary's queries is correct (since we don't know the discrete log of  $\hat{m}$ ). We will resolve the problem by guessing which query is correct: If  $b = 0$ , our guess is that none of them are correct. If  $b = 1$ , then our guess is that the  $d_2$ th query is correct.

Proceed as follows: Set  $\hat{\mu} = \mu$ . Consider all the  $H_3$  queries of form  $(A, B, \hat{m}, \hat{\mu}, \cdot, \pi^*)$ . If  $b = 1$  and there have been  $d_2$  or more of such queries, then expose using the result of the  $d_2$ th query. Otherwise, do a *start session* with the *dangling connect* connection assignment.

Note that we don't need to be concerned about the possibility of an initiator failure, if a bad event is about to occur (as mentioned at the beginning of the proof of this claim). It now easily follows that the above response is indistinguishable from the original simulator, if  $b$  and  $d_2$  are guessed correctly.

(c)  $H_3(A, B, X, \mu, \sigma, \pi)$ :

If  $\mu = Y g^{z_{i'j'}}$  for some  $(i', j')$  and

$$\sigma = Z \cdot X^{z_{i'j'} - r\alpha'[A, B, \pi]} \cdot g^{r^2 \cdot \alpha[A, B, \pi] \alpha'[A, B, \pi]} / \mu^{r\alpha[A, B, \pi]},$$

then stop the distinguisher and guess “True DH.”

Otherwise, if  $\hat{\mu}$  has been set,  $\mu = \hat{\mu}$ ,  $\pi = \pi^*$ ,  $b = 1$ , and this is the  $d_2$ th query of form  $H_3(A, B, \hat{m}, \hat{\mu}, \cdot, \pi^*)$ , then respond with the key of  $(\hat{i}, \hat{j})$  (note that the distinguisher is playing the ring master, and so has access to all the keys). Otherwise, return  $k \xleftarrow{R} \{0, 1\}^\kappa$ .

Note that we don’t have to be concerned with the possibility of an initiator failure, as above. It is then easy to see that the response is indistinguishable from the original simulator, as long as  $b$  and  $d_2$  have been guessed correctly.

4. If the simulation ends without outputting “True DH,” then output “Random.”

Note that if the adversary does make a bad query, we have probability at least  $1/T$  of guessing the correct initiator user instance, and probability at least  $1/2T$  of answering the  $A_2$  and  $H_3$  queries to that user instance correctly (thus allowing the DDH distinguisher to continue in a way that is indistinguishable from the regular simulator).

Now, the probability that the distinguisher  $D$  guesses correctly is at least

$$\begin{aligned} \Pr(D \text{ is correct}) &= \Pr(D \text{ guesses “DH True”} | \text{DH instance}) \Pr(\text{DH instance}) \\ &\quad + \Pr(D \text{ guesses “Random”} | \text{Random instance}) \Pr(\text{Random instance}) \\ &\geq \left(\frac{\epsilon}{2T^2}\right) \left(\frac{1}{2}\right) + \left(1 - \frac{T}{q}\right) \left(\frac{1}{2}\right), \end{aligned}$$

where the term  $T/q$  comes from the probability of the adversary “guessing” a random  $Z$  correctly on a random oracle query. Thus, the probability that  $D$  guesses correctly is at least  $\frac{1}{2} + \frac{\epsilon}{4T^2} - \frac{T}{2q}$ , which is non-negligibly more than  $\frac{1}{2}$ .  $\square$

## 3.9 Security of the PAK-X Protocol

### 3.9.1 The Simulator

The proof of simulatability of PAK-X is similar in structure to that of PAK. We name the actions of user instances on *deliver message* operations as follows:

**A0** Initiator instance action to start the protocol (i.e., an A0 action for an initiator instance is generating and sending  $m$ ).

**B1** Responder instance action upon receiving the first message (i.e.,  $m$ ).

**A2** Initiator instance action upon receiving the message from the responder (i.e.,  $(\mu, a, k)$ ).

**B3** Responder instance action upon receiving the second message from the initiator (i.e.,  $k'$ ).

We will let  $(i, j)$ ,  $(i', j')$ ,  $A$ ,  $B$ , and  $\pi^*$  have the same meaning as in the proof of PAK (see the beginning of Section 3.7.1). The notions of *acceptable partner* and *matching conversation* will also retain their meaning. For brevity, we will write  $H_0(\{A, B\}, \pi)$  to mean  $H_0(\min(A, B), \max(A, B), \pi)$ . In addition, we will write  $v^*$  and  $V^*$  to mean  $H_0(\{A, B\}, \pi^*)$  and  $g^{v^*}$ , respectively. (Note that when  $v^*$  and  $V^*$  appear in expressions for implicit oracle calls, their values might not yet be determined, i.e., the  $H_0$  queries may not yet have been made.)

The simulator will follow the same general rules as described in Section 3.7.1. Responses to *deliver message* and *random oracle* operations that do not fall under those rules, as well as to the new *get verifier* operation, are done as follows:

1.  $H_0(\{A, B\}, \pi)$

We presume that  $A$  and  $B$  are valid user IDs, say of users  $i$  and  $i'$  (otherwise, this query would be handled by one of the general rules, as mentioned in Section 3.7.1). Perform a *test password* on  $(i, i')$  (this is legal, since an *impl* operation will be made next to record the result of this oracle query). Now consider two cases:

**Case 1** No *get verifier* operation has yet been performed on  $(i, i')$ .

Generate and store  $v[\{A, B\}, \pi] \xleftarrow{R} \{0, 1\}^{|q|+\kappa}$ . Respond with  $v[\{A, B\}, \pi]$ .

This response is indistinguishable from the real system, since the only prior implicit query that could shadow this one is in *get verifier* on  $(i, i')$ .

However, this case assumes that no *get verifier* operation has yet been performed on  $(i, i')$ .

**Case 2** A *get verifier* operation has been performed on  $(i, i')$ .

In this case, the *test password* operation above has returned whether or not  $\pi = \pi^*$ . If  $\pi \neq \pi^*$ , then return  $v \xleftarrow{R} \{0, 1\}^{|q|+\kappa}$ . If  $\pi = \pi^*$  (in which case this query is shadowed), then return  $v[\{A, B\}]$  (which has been set while processing *get verifier*—see the description of *get verifier* below).

2.  $H'_0(A, B, c)$

Return  $u \xleftarrow{R} \{0, 1\}^{|q|+\kappa}$ .

3.  $H_1(A, B, V)$

Generate  $\alpha[A, B, \pi] \xleftarrow{R} Z_q$  and store it. Then generate  $h \xleftarrow{R} Z_p^*$  and  $\beta \xleftarrow{R} Z_{[2^\eta/p]}$ , and return  $(h^q g^{\alpha[A, B, \pi]} \bmod p) + \beta p$ . Note that this will be indistinguishable from a random bit string of length  $\eta$ , since  $h^q g^{\alpha[A, B, \pi]} \bmod p$  is a random element from  $Z_p^{*11}$  and  $\frac{2^\eta \bmod p}{2^\eta}$  is negligible.

4.  $H_{2a}(A, B, m, \mu, a, \sigma, \tau, V)$

**Case 1** Some prior B1 query has recorded a tuple of the form  $(i', j', m, \mu, y_{i'j'}, c_{i'j'}, k_{i'j'})$ , with  $A = PID_{i'j'}$  and  $B = ID_{i'}$ , for some values of  $y_{i'j'}$ ,  $c_{i'j'}$ , and  $k_{i'j'}$ . This implies that a B1( $m$ ) query was made to  $(i', j')$  and returned  $(\mu, g^{H'_0(A, B, c_{i'j'})}, k_{i'j'})$ , but no A0 query to an acceptable partner returned  $m$ —see the B1 query description below. (In other words, this  $H_{2a}$  query might be shadowed by the implicit  $H_{2a}$  query from case 2 of the B1 action.)

---

<sup>11</sup>To see this, note that  $h^q$  is a random element from the subgroup of order  $r$  in  $Z_p^*$  and  $g^{\alpha[A, B, \pi]}$  is a random element of the subgroup of order  $q$  in  $Z_p^*$ .

W.o.p., there will be at most one such tuple, since  $\mu$  values stored in these tuples are random and independent. Consider two cases:

**Case 1a** Query  $H_1(A, B, V)$  has been asked,  $(\frac{m}{(H_1(A, B, V))^r})^{y_{i'j'}} = \sigma$ , and  $\tau = V^{H'_0(A, B, c_{i'j'})}$ .

The simulator proceeds as follows: If there has been a successful guess  $\pi^*$  on  $\{A, B\}$  and  $V = g^{H_0(\{A, B\}, \pi^*)}$ , then call this  $H_{2a}$  query a *successful  $H_{2a}$  query on  $(i', j')$* .

If there hasn't been a successful guess on  $\{A, B\}$ , then check all the  $H_0$  queries, and look for  $\pi$  that results in  $V = g^{H_0(\{A, B\}, \pi)}$ . If such a  $\pi$  is found (there can be at most one such  $\pi$ , w.o.p., by the randomness of  $H_0$ ), and there never was an unsuccessful guess on  $\{A, B\}$  for  $\pi$ :

- (a) If a *test instance password* operation has not previously been performed on  $(i', j')$ , perform a *test instance password* operation with arguments  $(i', j')$  and  $\pi$ . If the test is successful, we also call this query a *successful  $H_{2a}$  query on  $(i', j')$* .
- (b) If a *test instance password* operation has previously been performed on  $(i', j')$ , then abort. We will call this event *an  $H_{2a}$  failure*.

In explanation, let's note that  $V$  is independent of  $V^*$ , unless either the discrete log of  $V$  has been returned by a prior  $H_0$  query, or  $V$  has been returned by a *get verifier* query (in which case  $V = V^*$ ). The following claim shows that even in the latter case, this  $H_{2a}$  query can't be "correct" unless a prior  $H_0$  query resulted in the discrete log of  $V$  (or else there has been a successful guess on  $\{A, B\}$ ):

**Claim 9.** *Let  $(\mu, a, k)$  be returned by a  $B1(m)$  query to  $(i', j')$ . Let  $A = PID_{i'j'}$  and  $B = ID_{i'}$ . Suppose *get verifier* is performed on  $\{A, B\}$  (either before or after the  $B1$  action), and returns  $V^*$ . Then, w.o.p., no query*

$$H_{2a}(A, B, m', \mu', a, \text{DH}(\mu', \frac{m'}{(H_1(A, B, V^*))^r}), \text{DH}(a, V^*), V^*),$$

for any  $m'$  and  $\mu'$ , will be made, unless by the time of the  $H_{2a}$  query there has been either a successful guess on  $\{A, B\}$ , or an  $H_0(\{A, B\}, \pi)$  query, for some  $\pi$ , with  $V^* = g^{H_0(\{A, B\}, \pi)}$ .

The proof of the claim appears in Section 3.9.2.

**Case 1b** Otherwise.

Do nothing.

Finally, if this query is a successful  $H_{2a}$  query on  $(i', j')$  for some  $(i', j')$ , then return  $k_{i'j'}$ . Otherwise, return  $k \xleftarrow{R} \{0, 1\}^\kappa$ .

If no  $H_{2a}$  failure occurs as a result of this  $H_{2a}$  query, then the simulation will be indistinguishable from the real world, as follows: In the real world the  $k$  value sent by  $(i', j')$  is

$$H_{2a}(A, B, m, \mu, g^{H'_0(A, B, c_{i'j'})}, \text{DH}(\mu, \frac{m}{(H_1(A, B, V^*))^r}, (V^*)^{H'_0(A, B, c_{i'j'})}, V^*).$$

Therefore, if this  $H_{2a}$  query is successful, it will return a value consistent with the adversary's view. On the other hand, if this query is not successful, then generating a random response is indistinguishable from the adversary's view in the real world.

The following claim shows that w.o.p. there will be no failure:

**Claim 10.** *W.o.p., no  $H_{2a}$  failure will occur.*

The proof appears in Section 3.9.2.

As can be seen, an  $H_{2a}$  query is the only place in the simulation that a *test instance password* query can be made on a responder instance. We can therefore state the following claim (similarly to PAK and PPK):

**Claim 11.** *If a test instance password query is made on a responder instance  $(i', j')$ , then  $(i', j')$  has not had a matching conversation with an acceptable partner.*

The reason is that an  $H_{2a}$  query can result in a *test instance password*

query on  $(i', j')$  only if a tuple  $(i', j', m, \mu, y_{i'j'}, c_{i'j'}, k_{i'j'})$  has previously been recorded. It will become apparent below (see the B1 query description) that, w.o.p., this could only happen if  $(i', j')$  has not had a matching conversation with an acceptable partner.

**Case 2** No tuple of the form  $(i', j', m, \mu, y_{i'j'}, c_{i'j'}, k_{i'j'})$  has been recorded with  $A = PID_{i'j'}$  and  $B = ID_{i'}$ .

Return  $k \xleftarrow{R} \{0, 1\}^\kappa$ . The only way this response could be distinguishable from the real system is if this query would be shadowed by an implicit  $H_{2a}$  query from case 1 of some B1 action, i.e., if  $m$  and  $\mu$  are from a matching conversation of two valid user instances, and  $\sigma = \text{DH}(\mu, \frac{m}{(H_1(A, B, V))^r})$ . However, by the following claim, w.o.p. this will not occur.

**Claim 12.** *Let  $m$  be returned by an A0 query to  $(i, j)$  with  $A = ID_i$  and  $B = PID_{ij}$ , and  $\mu$  be returned by a subsequent B1( $m$ ) query to  $(i', j')$  with  $B = ID_{i'}$  and  $A = PID_{i'j'}$ . Then, w.o.p., there will never be (neither before nor after the A0 and B1 queries) an oracle query to either  $H_{2a}$ ,  $H_{2b}$ , or  $H_3$ , that includes  $(A, B, m, \mu, \text{DH}(\mu, \frac{m}{(H_1(A, B, V))^r}), V)$ , for any  $V$ .*

This claim is proved in the same way as Claim 3 (with obvious changes to accommodate the differences between PAK and PAK-X).

5.  $H_{2b}(A, B, m, \mu, \sigma, a, k, c, V)$

Return  $k' \xleftarrow{R} \{0, 1\}^\kappa$ .

This is indistinguishable from the real system by the following argument: The only implicit  $H_{2b}$  query that could shadow this one is the query from case 1 of an A2 action. However, that shadowing is, w.o.p., impossible by Claim 12. (Note that the  $H_{2b}$  queries in case 2 of the A2 action are explicit, i.e., all of their arguments are known, and so a query-response pair would be stored for them.)

6.  $H_3(A, B, m, \mu, \sigma, c, V)$

Return  $K \xleftarrow{R} \{0, 1\}^\kappa$ .

As for  $H_{2b}$  queries, indistinguishability easily follows by Claim 12.

7. A0 query to  $(i, j)$

Generate and store  $w \xleftarrow{R} Z_q$ , and send  $m = g^w$ . Clearly,  $m$  is uniformly drawn from  $G_{p,q}$ , just as in the real system.

8. B1( $m$ ) query to  $(i', j')$

Generate  $y_{i'j'} \xleftarrow{R} Z_q$ ,  $c_{i'j'} \xleftarrow{R} \{0, 1\}^\kappa$ ,  $k_{i'j'} \xleftarrow{R} \{0, 1\}^\kappa$ . Send  $\mu = g^{y_{i'j'}}$ ,  $a = g^{H'_0(A, B, c_{i'j'})}$ , and  $k = k_{i'j'}$ . Now consider two cases:

**Case 1** The value of  $m$  has been sent by an acceptable partner.

The simulation corresponds to the implicit oracle query

$$k_{i'j'} \oplus c_{i'j'} = H_{2a}(A, B, m, \mu, a, \text{DH}(\mu, \frac{m}{(H_1(A, B, V^*))^r}), (V^*)^{H'_0(A, B, c_{i'j'})}, V^*).$$

Since  $\mu$  is a freshly chosen random number, this implicit query, w.o.p., will not shadow any prior  $H_{2a}$  queries (whether explicit or implicit). Consequently,  $k_{i'j'}$  will be indistinguishable from the value sent in the real system.

**Case 2** The value of  $m$  has not been sent by an acceptable partner.

In this case, record the tuple  $(i', j', m, \mu, y_{i'j'}, c_{i'j'}, k_{i'j'})$ . It is important to note (for Claim 11), that if this tuple is recorded, then, w.o.p.,  $(i', j')$  will never be considered to have a matching conversation with an acceptable partner, due to the fact that values of  $m$  are generated randomly by initiator instances. (Recall that in our definition of “matching conversation” we are not concerned with the timing of sends and receives.)

As in the previous case, we have an implicit oracle query

$$k_{i'j'} \oplus c_{i'j'} = H_{2a}(A, B, m, \mu, a, \text{DH}(\mu, \frac{m}{(H_1(A, B, V^*))^r}), (V^*)^{H'_0(A, B, c_{i'j'})}, V^*),$$

which is indistinguishable as above by the randomness of  $\mu$  values.

9. A2( $\mu, a, k$ ) query to  $(i, j)$

**Case 1**  $(i, j)$  has had a matching conversation with an acceptable partner  $(i', j')$ .

Generate  $k'_{ij} \xleftarrow{R} \{0, 1\}^\kappa$ , send  $k' = k'_{ij}$ , set status to Accept, and perform a *start session* operation with the connection assignment *open for connection from*  $(i', j')$ . This corresponds to the following implicit oracle queries:

$$k'_{ij} = H_{2b}(A, B, m, \mu, \text{DH}(\mu, \frac{m}{(H_1(A, B, V^*))^r}), a, k'_{i'j'}, c_{i'j'}, V^*),$$

$$K_{ij} = H_3(A, B, m, \mu, \text{DH}(\mu, \frac{m}{(H_1(A, B, V^*))^r}), c_{i'j'}, V^*),$$

where  $K_{ij}$  is the random session key assigned by the ring master at the time of the *start session* operation.

By Claim 12, these implicit queries couldn't shadow any past or future explicit queries. It is also easy to see that these implicit queries will not shadow any other implicit queries, unless there is a collision of  $m$  values between two initiator instances (and that only occurs with negligible probability).

The *open* connection assignment will be legal, since the only place that the simulator could perform a *test instance password* operation on  $(i, j)$  would be in case 2 of the A2 query (see there). Also, w.o.p., we will never have two initiator instances  $(i_1, j_1)$  and  $(i_2, j_2)$  that are open for connection from the same responder instance  $(i', j')$ , since that would imply that  $(i_1, j_1)$  and  $(i_2, j_2)$  generated the same  $m$  value (otherwise, they couldn't both have had matching conversations with  $(i', j')$ ).

**Case 2**  $(i, j)$  has not had a matching conversation with an acceptable partner.

Look for a value of  $c$  for which an  $H'_0(A, B, c)$  query has been made and  $a = g^{H'_0(A, B, c)}$ . (W.o.p. there will be at most one such value of  $c$ , due to

the randomness of  $H'_0$ .) If no such  $c$  is found, then reject. This is indistinguishable from the real system, since w.o.p. the  $a$  received would not pass the test  $a = g^{H'_0(A,B,c)}$ , no matter what  $c$  would have been computed in the real system.

Otherwise (if such a  $c$  is found), look for a value of  $V$  for which an  $H_1(A, B, V)$  query has been made, and another query

$$H_{2a}(A, B, m, \mu, a, \mu^{w-r\alpha[A,B,V]}, V^{H'_0(A,B,c)}, V)$$

has been made and returned  $k \oplus c$ . (W.o.p. there will be at most one such value of  $V$ , due to the randomness of  $H_{2a}$ .) If no such  $V$  is found, then reject. This is indistinguishable from the real system, since w.o.p. the  $k$  received would be incorrect in the real system (i.e., the correct  $k = c \oplus H_{2a}(A, B, m, \mu, a, \text{DH}(\mu, \frac{m}{(H_1(A,B,V^*))^r}), (V^*)^{H'_0(A,B,c)}, V^*))$  would be independent of the adversary's view).

Suppose such a  $V$  is found. Consider two cases:

**Case 2a** A *get verifier* query has been performed on  $(i, i')$  and returned  $V^*$ .

If  $V \neq V^*$ , then reject. This is clearly indistinguishable from the real system, since the correct  $k$  is independent of the adversary's view (as the  $V$  argument to  $H_{2a}$  is incorrect).

If  $V = V^*$ , then proceed as follows:

- (a) Set  $k' = H_{2b}(A, B, m, \mu, \mu^{w-r\alpha[A,B,V]}, a, k, c, V)$ .
- (b) Send  $k'$ .
- (c) Accept.
- (d) Set  $K = H_3(A, B, m, \mu, \mu^{w-r\alpha[A,B,V]}, c, V)$ .
- (e) Expose using session key  $K$  (this is allowed, since  $(i, j)$  is a client instance, and a *get verifier* query has been performed on  $(i, i')$ ).

Note that the values of  $k'$  and  $K$  are computed through explicit oracle queries (we can think of these as subroutine calls within the simulator),

and the queries and responses are recorded, as usual. It is clear that the values of  $k'$  and  $K$  produced in this case are the same as would be computed in the real system.

**Case 2b** No *get verifier* query has been performed on  $(i, i')$ .

Look for a value of  $\pi$  such that  $H_0(\{A, B\}, \pi)$  has been asked and  $V = g^{H_0(\{A, B\}, \pi)}$ . (W.o.p. there will be at most one such value of  $\pi$ , due to the randomness of  $H_0$ .) If no such  $\pi$  is found, then reject. This is indistinguishable from the real system, since w.o.p. the  $k$  received would be incorrect in the real system (as the  $V$  argument to  $H_{2a}$  is incorrect w.o.p.).

Otherwise (if such a  $\pi$  is found), perform a *test instance password* operation with arguments  $(i, j)$  and  $\pi$  (note that this is the only place where we could perform this operation for an initiator instance, and no session has been started yet, so the operation is legal). If the guess is not successful, then reject (this is indistinguishable from the real system, by an argument similar to the one above). If the guess is successful, then:

- (a) Set  $k' = H_{2b}(A, B, m, \mu, \mu^{w-r\alpha[A, B, V]}, a, k, c, V)$ .
- (b) Send  $k'$ .
- (c) Accept.
- (d) Set  $K = H_3(A, B, m, \mu, \mu^{w-r\alpha[A, B, V]}, c, V)$ .
- (e) Expose using session key  $K$  (this is allowed, since there was a successful guess on the password).

This is correct similarly to above.

10. B3( $k'$ ) query to  $(i', j')$

**Case 1**  $(i', j')$  has had a matching conversation with an acceptable partner  $(i, j)$ .

Set status to Accept and perform a *start session* operation with a *connect to*  $(i, j)$  connection assignment. This is legal because

- (a) w.o.p., the instance  $(i, j)$  will still be open for connection by the randomness of  $\mu$  values sent by responder instances (so that, w.o.p., for each initiator  $(i, j)$ , there will be at most one  $(i', j')$  with which it has had a matching conversation, and so at most one instance will try to connect to  $(i, j)$ ), and
- (b) by Claim 11, there could not have been a *test instance password* query on  $(i', j')$ , since  $(i', j')$  has had a matching conversation with an acceptable partner.

**Case 2** The current value  $m$  has been sent by some acceptable partner  $(i, j)$ , and  $(\mu, a, k_{i'j'})$  have been received by  $(i, j)$ , but the value of  $k'$  that was received has not been sent by  $(i, j)$ .

Reject. This is indistinguishable from the real system since  $k'$  is invalid.

**Case 3** The current value  $m$  has been sent by some acceptable partner  $(i, j)$ , but the triple received by  $(i, j)$  in A2 is not equal to  $(\mu, a, k_{i'j'})$  sent by  $(i', j')$ .

Reject. By Claim 12, w.o.p., the  $H_{2b}$  query that would produce the correct value of  $k'$  has never been asked. Also, no implicit query could have produced the correct  $k'$  (note that  $(i, j)$  did not have a matching conversation, and thus did not “make” an implicit  $H_{2b}$  query; also, any other initiator instance would, w.o.p., have a different value of  $m$ ). Thus, the correct value of  $k'$  is independent of the adversary’s view, and so it is safe to reject.

**Case 4** Otherwise. (Note that in this case, the current value of  $m$  has not been sent by an acceptable partner, and thus a tuple must have been recorded by  $(i', j')$  in case 2 of the B1 action.)

Look for a value of  $V$  such that an  $H_1(A, B, V)$  query has been asked and a query  $H_{2b}(A, B, m, \mu, (\frac{m}{(H_1(A, B, V))^r})^{y_{i'j'}}, a, k_{i'j'}, c_{i'j'}, V)$  has been asked and returned  $k'$ . (W.o.p. there will be at most one such value of  $V$ , due to the randomness of  $H_{2b}$ .) If no such  $V$  is found, then reject. This is

indistinguishable from the real system, since w.o.p. the  $k'$  received would be incorrect in the real system (i.e., the correct  $k'$  would be independent of the adversary's view). (The only way  $k'$  could be correct, other than through a random guess, is if it is the result of an implicit  $H_{2b}$  query in case 1 of A2. However, it is easy to see that if that was the case, we couldn't get to case 3 of B3.)

Otherwise (if such a  $V$  is found), consider two cases:

**Case 4a** There has been a successful guess  $\pi$  on  $\{A, B\}$ .

If  $V = g^{H_0(\{A, B\}, \pi)}$ , then expose with key

$$K = H_3(A, B, m, \mu, (\frac{m}{(H_1(A, B, V))^r})^{y_{i'j'}}, c_{i'j'}, V).$$

This is clearly allowed and indistinguishable from the real system.

Otherwise, reject. This is clearly indistinguishable from the real system, since the correct  $k'$  is independent of the adversary's view (as the  $V$  argument to  $H_{2a}$  is incorrect).

**Case 4b** There has not been a successful guess on  $\{A, B\}$ .

In this case, reject.

Let's show that this behavior is indistinguishable from the real system. Suppose we were instead supposed to accept. The value  $c_{i'j'}$  is independent of the adversary's view, unless the adversary has queried  $H_{2a}(A, B, m, \mu, a, \text{DH}(\mu, \frac{m}{(H_1(A, B, V^*))^r}), \text{DH}(a, V^*), V^*)$ . Since the value of  $c_{i'j'}$  was present in one of the adversary's  $H_{2b}$  queries, it follows that, w.o.p., the adversary has indeed made the above  $H_{2a}$  query. Now, from Claim 9 and the discussion preceding that claim, it follows that, w.o.p., the adversary has made a prior query  $H_0(\{A, B\}, \pi)$ , for some  $\pi$ , such that  $V^* = g^{H_0(\{A, B\}, \pi)}$ . However, that would imply that  $H_0(\{A, B\}, \pi) = v^*$ . It is easy to see that, w.o.p.,  $H_0(\{A, B\}, \pi)$  would only return  $v^*$  if  $\pi = \pi^*$ . From the  $H_{2a}$  action description above, it is easy to see that this situation would, w.o.p., result in a successful

guess on  $\{A, B\}$  (made during the  $H_{2a}$  query). Thus, there has already been a successful guess on  $\{A, B\}$ , which contradicts our original assumption.

11. *get verifier* on user pair  $(i, i')$

Let  $A$  and  $B$  be the IDs of users  $i$  and  $i'$ , respectively. Send a *get verifier* query on  $(i, i')$  to the ring master (i.e., in the ideal world). The ring master will return whether any prior *test password* query on  $(i, i')$  was successful. If there was a successful guess on  $\{A, B\}$  for some  $\pi$  (whether through *test instance password* or *test password*), then return  $g^{H_0(\{A, B\}, \pi)}$ .

Otherwise (i.e., if no query was successful), generate and store  $v[\{A, B\}] \xleftarrow{R} \{0, 1\}^{|\mathcal{Q}| + \kappa}$ . Return  $g^{v[\{A, B\}]}$ . This corresponds to the implicit oracle query

$$v[\{A, B\}] = H_0(\{A, B\}, \pi^*).$$

Clearly, this implicit query cannot shadow any prior explicit  $H_0$  query, or else some prior *test password* would have been successful.

### 3.9.2 Proofs of Claims

*Proof of Claim 9.* Suppose that with some nonnegligible probability  $\epsilon$  there will be some responder instance  $(\hat{i}', \hat{j}')$  (with  $B = ID_{\hat{i}'}$  and  $A = PID_{\hat{i}'\hat{j}'}$ ) such that the following “bad event” occurs:

1. query  $B1(\hat{m})$  is made to  $(\hat{i}', \hat{j}')$  and returns  $(\hat{\mu}, \hat{a}, \hat{k})$ ,
2. *get verifier* is performed on  $\{A, B\}$  and returns  $V^*$  (either before or after the  $B1$  action), and
3. query

$$H_{2a}(A, B, m, \mu, \hat{a}, \text{DH}(\mu, \frac{m}{(H_1(A, B, V^*))^r}), \text{DH}(\hat{a}, V^*), V^*)$$

(for some  $m$  and  $\mu$ ) is made, before there has been either a successful guess on  $\{A, B\}$  or an  $H_0(\{A, B\}, \pi)$  query, for some  $\pi$ , with  $V^* = g^{H_0(\{A, B\}, \pi)}$ .

We will then construct a distinguisher  $D$  for DDH. Let  $(X, Y, Z)$  be the challenge DDH instance.

1. Generate random  $d$  between 1 and  $T$ .
2. We will be running the simulator in the normal manner, but playing the ring master also. (That is, we will choose passwords for user pairs, and answer any test password queries.) Run the simulation until the  $d$ th pair of users  $(A, B)$  is mentioned. (This may be from a *get verifier* query on users with IDs  $A$  and  $B$ , or from an *initialize user instance* query with user ID  $A$  and partner ID  $B$ , or vice-versa.) If we guessed  $d$  correctly, this pair will be the identities of the users in the “bad event.”
3. Once  $A$  and  $B$  are set, continue as in the original simulator, except:

- (a) *get verifier* on users with IDs  $A$  and  $B$ :

Respond with  $X$ . Note that this results in  $V^* = X$ .

If the “bad event” is about to occur, then this response is correct, since there could not have been a successful guess on  $\{A, B\}$ .

- (b)  $B1(m)$  query to instance  $(i', j')$  with  $ID_{i'} = B$  and  $PID_{i'j'} = A$ :

Generate  $y_{i'j'} \xleftarrow{R} Z_q$ ,  $z_{i'j'} \xleftarrow{R} Z_q$ , and  $k_{i'j'} \xleftarrow{R} \{0, 1\}^\kappa$ . Send  $\mu = g^{y_{i'j'}}$ ,  $a = Y g^{z_{i'j'}}$ , and  $k = k_{i'j'}$ .

Note that this corresponds to an implicit query

$$y + z_{i'j'} = H'_0 \left( A, B, k_{i'j'} \oplus H_{2a} \left( A, B, m, \mu, Y g^{z_{i'j'}} \right), \right. \\ \left. \text{DH}(\mu, \frac{m}{(H_1(A, B, X))^r}), \text{DH}(Y g^{z_{i'j'}}, X), X \right).$$

W.o.p., this implicit query will not shadow any prior  $H'_0$  query (whether implicit or explicit), by the randomness of  $k_{i'j'}$ .

- (c)  $H_0(\{A, B\}, \pi)$  query: Generate and store  $v[\{A, B\}, \pi] \xleftarrow{R} \{0, 1\}^{|\mathcal{Q}|+\kappa}$ . Respond with  $v[\{A, B\}, \pi]$ .

If the “bad event” is about to occur and  $d$  has been guessed correctly, then this response is appropriate, as the “bad event” rules out  $H_0$  queries that would return the discrete log of  $V^*$ .

- (d)  $H'_0(A, B, c)$  query: Behave as in the original simulator.

The only way this behavior could be distinguishable from the original simulator is if this query was shadowed by an implicit query from a B1 action. However, it is easy to see that this will not occur, w.o.p., unless the adversary first makes the query

$$H_{2a}(A, B, m, \mu, Yg^{z_{i'j'}}, \text{DH}(\mu, \frac{m}{(H_1(A, B, X))^r}), \text{DH}(Yg^{z_{i'j'}}, X), X),$$

(since without such an  $H_{2a}$  query, the value of  $c$  that would cause the shadowing would be independent of the adversary’s and simulator’s view). However, such an  $H_{2a}$  query would imply that the bad event has already occurred, in which case we are no longer concerned with indistinguishability.

- (e)  $A2(\mu, a, k)$  query to instance  $(i, j)$  with  $ID_i = A$  and  $PID_{ij} = B$ , not part of a matching conversation: If the value of  $a$  is not equal to  $Yg^{z_{i'j'}}$  for any  $(i', j')$ , then proceed as in the original simulator. Otherwise, reject.

In the first case, the behavior is clearly indistinguishable. On the other hand, suppose  $a = Yg^{z_{i'j'}}$  for some  $(i', j')$ . Our response could be incorrect if query

$$H_{2a}(A, B, m, \mu, Yg^{z_{i'j'}}, \text{DH}(\mu, \frac{m}{(H_1(A, B, X))^r}), \text{DH}(Yg^{z_{i'j'}}, X), X)$$

has been made. However, that would imply that the bad event has already occurred, in which case we are no longer concerned with indistinguishability.

- (f)  $B3(k')$  query to responder instance  $(i', j')$ , where  $PID_{i'j'} = A$  and  $ID_{i'} = B$ : If there is no matching conversation with an acceptable partner, then respond with reject. Otherwise, behave as in the original simulator.

If the “bad event” is about to occur and  $d$  has been guessed correctly, then this response is appropriate: If there was no matching conversation with an acceptable partner and this query was supposed to result in an accept, then there would be a successful guess on  $\{A, B\}$ , and that would contradict the definition of the “bad event” (unless the “bad event” has already occurred, in which case we don’t care whether or not the response was correct).

- (g)  $H_{2a}(A, B, m, \mu, a, \sigma, \tau, X)$  query where  $a = Yg^{z_{i'j'}}$  for some  $(i', j')$  and  $\tau = ZX^{z_{i'j'}}$ : Stop the distinguisher and guess “True DH.”

4. If the simulation ends without outputting “True DH,” then output “Random.”

Note that if the “bad event” does occur, we have probability at least  $1/T$  of guessing the correct pair of users. Now, the probability that the distinguisher  $D$  guesses correctly is at least

$$\begin{aligned} \Pr(D \text{ is correct}) &= \Pr(D \text{ guesses “DH True”} | \text{DH instance}) \Pr(\text{DH instance}) \\ &\quad + \Pr(D \text{ guesses “Random”} | \text{Random instance}) \Pr(\text{Random instance}) \\ &\geq \left(\frac{\epsilon}{T}\right) \left(\frac{1}{2}\right) + \left(1 - \frac{T}{q}\right) \left(\frac{1}{2}\right), \end{aligned}$$

where the term  $T/q$  comes from the probability of the adversary “guessing” a random  $Z$  correctly on a random oracle query. Thus, the probability that  $D$  guesses correctly is at least  $\frac{1}{2} + \frac{\epsilon}{2T} - \frac{T}{2q}$ , which is non-negligibly more than  $\frac{1}{2}$ .  $\square$

*Proof of Claim 10.* We will call an oracle query of form  $H_{2a}(A, B, m, \mu, a, \sigma, \tau, V)$  “bad” if

1. some responder instance  $(i', j')$ , with  $ID_{i'} = B$  and  $PID_{i'j'} = A$ , has sent  $\mu$  (w.o.p., by randomness of  $\mu$  values, there will be at most one such instance),

2.  $\sigma = \text{DH}(\mu, \frac{m}{(H_1(A, B, V))^r})$ , and
3.  $\tau = \text{DH}(a, V)$ .

Suppose that with some nonnegligible probability  $\epsilon$  there will be an  $H_{2a}$  failure. This implies that there will be some responder instance  $(\hat{i}', \hat{j}')$  (with  $B = ID_{\hat{i}'}$  and  $A = PID_{\hat{i}'\hat{j}'}$ ) such that the following “bad event” occurs:

1. query  $\text{B1}(\hat{m})$  is made to  $(\hat{i}', \hat{j}')$  and returns  $(\hat{\mu}, \hat{a}, \hat{k})$ , and
2. at least two “bad” queries are made with  $(A, B, \hat{m}, \hat{\mu})$  and distinct values of  $V$ , before there is a successful guess on  $\{A, B\}$ .

(Note that we speak of distinct values of  $V$ , since, w.o.p., distinct values of  $\pi$  will result in distinct values of  $\pi$ .) We will then show how to construct a distinguisher  $D$  for the DDH problem.

The idea of our distinguisher  $D$ , with input  $(X, Y, Z)$ , is to simulate the original simulator, incorporating  $X$  and  $Y$  into some of the responses. Then, if  $D$  runs indistinguishably from the simulator up to the bad event, we will be able to use the logs to determine whether or not  $Z = \text{DH}(X, Y)$ .

Our distinguisher  $D$  for input  $(X, Y, Z)$  runs as follows:

1. Generate random  $d$  between 1 and  $T$ .
2. Initialize two lists  $\text{BAD}_0$  and  $\text{BAD}_1$  (initially empty).
3. We will be running the simulator in the normal manner, but playing the ring master also. (That is, we will choose passwords for user pairs, and answer any test password queries.) Run the simulation until the  $d$ th pair of users  $(A, B)$  is mentioned. (This may be from an oracle query  $H_1(A, B, \cdot)$ , or from an *initialize user instance* query with user ID  $A$  and partner ID  $B$ , or vice-versa.) If we guessed  $d$  correctly, this pair will be the identities of the users in the “bad event.”

We will say that  $V^*$  is *determined* when either the query  $H_0(\{A, B\}, \pi^*)$  or the query *get verifier* on  $\{A, B\}$  is made. Note that, since the distinguisher itself

generates  $\pi^*$ , we will be able to detect whether or not  $V^*$  has been determined, and we will also be able to compute  $V^*$  after that point.

4. Once  $A$  and  $B$  are set, continue as in the original simulator, except:

- (a)  $B1(m)$  query to instance  $(i', j')$  with  $ID_{i'} = B$  and  $PID_{i'j'} = A$ : generate  $z_{i'j'} \xleftarrow{R} Z_q$ , set  $\mu = Yg^{z_{i'j'}}$ , and then proceed as in the original simulator.
- (b)  $H_1(A, B, V)$ : If  $V^*$  has been determined and  $V = V^*$ , then set  $b[V] = b[V^*] = 0$ . (Note that w.o.p. the query  $H_1(A, B, V^*)$  will not be made until after  $V^*$  has been determined.) Otherwise, let  $b[V] \xleftarrow{R} \{0, 1\}$ . Respond with  $(X^{b[V]} \cdot h^q g^{\alpha[A, B, V]} \bmod p) + \beta p$ , for  $h \xleftarrow{R} Z_p^*$ ,  $\alpha[A, B, V] \xleftarrow{R} Z_{p-1}$ , and  $\beta \xleftarrow{R} Z_{\lfloor 2^n/p \rfloor}$ .
- (c)  $A2(\mu, a, k)$  query to initiator instance  $(i, j)$ , where  $ID_i = A$  and  $PID_{ij} = B$ : Behave as in the original simulator, except if we get into case 2, then only check oracle queries with  $V = V^*$  (and if  $V^*$  has not yet been determined, then simply reject).

Note that  $(H_1(A, B, V^*))^r = g^{r \cdot \alpha[A, B, V^]}$ , since  $b[V^*] = 0$ . Thus, the original simulator will work correctly in this case. It is safe to ignore oracle queries with  $V \neq V^*$ , since those wouldn't lead to an accept (as the *test instance password* query would fail).

- (d)  $B3(k')$  query to responder instance  $(i', j')$ , where  $PID_{i'j'} = A$  and  $ID_{i'} = B$ : If there is no matching conversation with an acceptable partner, then respond with reject. Otherwise, behave as in the original simulator.

If the “bad event” is about to occur and  $d$  has been guessed correctly, then this response is appropriate: If this query was supposed to result in an accept, then there would be a successful guess on  $\{A, B\}$  before the second “bad” oracle query, and that would contradict the definition of the “bad event.” On the other hand, if the “bad event” has already occurred, then we don't care whether or not the response was correct.

- (e)  $H_{2a}(A, B, m, \mu, a, \sigma, \tau, V)$  query:

If  $H_1(A, B, V)$  was queried and there is an instance  $(i', j')$  with  $B = ID_{i'}$  that was queried with  $B1(m)$  and returned  $\mu = Yg^{z_{i'j'}}$ , then do the following (and otherwise, answer the query as in the original simulator):

First note that if this is a bad query, then

$$\sigma = DH(\mu, \frac{m}{(H_1(A, B, V))^r}) = DH(Yg^{z_{i'j'}}, \frac{m}{X^{b[V]r}g^{r \cdot \alpha[A, B, V]}}).$$

Now compute

$$\gamma = \sigma m^{-z_{i'j'}} X^{b[V]r z_{i'j'}} Z^{b[V]r} Y^{r \alpha[A, B, V]} g^{r \cdot \alpha[A, B, V] z_{i'j'}}.$$

Put  $\gamma$  on the list  $BAD_{b[V]}$ . Then respond with a random  $k$ .

If the bad event is about to occur, then this response will be indistinguishable from the original simulator, since no successful guess on  $\{A, B\}$  can occur before the bad event.

At the end of the simulation, check whether the lists  $BAD_0$  and  $BAD_1$  intersect (note that this check can be done in time  $O(T \log T)$ , by sorting the lists together). If yes, then output “True DH,” otherwise output “Random.”

Note that the values stored in  $BAD_0$  and  $BAD_1$  are simply guesses of  $DH(m, Y)$ , assuming  $Z = DH(X, Y)$ . If  $Z$  is random, then the probability of an intersection between the lists would be at most  $T^2/q$  by the union bound, since  $Z^r$  would be a random element of  $G_{p,q}$  (note that  $q$  and  $r$  are relatively prime, and  $m \neq 0 \bmod p$  because of the test by  $B$ ).

On the other hand, suppose  $Z = DH(X, Y)$ . If the adversary makes two “bad” queries for the pair of users  $(A, B)$ , for values  $V_1, V_2$  with  $b[V_1] \neq b[V_2]$ , then the distinguisher will correctly answer “True DH” (since each “bad” query will result in  $\gamma = DH(m, Y)$ ). The probability of the “bad event” is  $\epsilon$ . The probability of guessing  $d$  correctly is  $\frac{1}{T}$ . The probability of  $b[V_1] \neq b[V_2]$  for  $\pi_1 \neq \pi_2$  is  $\frac{1}{2}$ . All of these events are independent.

Now, the probability that the distinguisher  $D$  guesses correctly is at least

$$\begin{aligned} \Pr(D \text{ is correct}) &= \Pr(D \text{ guesses "DH True"} | \text{DH instance}) \Pr(\text{DH instance}) \\ &\quad + \Pr(D \text{ guesses "Random"} | \text{Random instance}) \Pr(\text{Random instance}) \\ &\geq \left(\frac{\epsilon}{2T}\right) \left(\frac{1}{2}\right) + \left(1 - \frac{T^2}{q}\right) \left(\frac{1}{2}\right). \end{aligned}$$

Thus the probability that  $D$  is correct is at least  $\frac{1}{2} + \frac{\epsilon}{4T} - \frac{T^2}{2q}$ , which is non-negligibly more than  $\frac{1}{2}$ .  $\square$

### 3.10 Conclusions and Open Problems

We have presented new formal definitions of security for password-authenticated key exchange protocols, with several variations: explicit authentication, implicit authentication, and resilience to server compromise. For each variation of the model, we have presented a new, efficient, and provably secure construction: the PAK, PPK, and PAK-X protocols. The proofs of security are based on the DDH problem and the random oracle assumption.

The major open problem seems to be the construction of provably secure password-authenticated key exchange protocols without using random oracles. It seems unlikely that random oracles can be easily removed from our schemes, as the proofs rely on the random oracle model quite heavily. One of the major issues in proving the security of a protocol in our security model is the ability of the simulator to extract the adversary's attempted guess of the password (as passwords appear explicitly in the ideal world). In our proofs, this extraction is accomplished by examining the adversary's random oracle queries, which is clearly impossible in a standard security model. It may, however, be possible to construct protocols based on zero-knowledge techniques, where the simulator would extract attempted passwords through rewinding. The problem with this approach is that the best known zero-knowledge protocols in the general concurrent setting take a polynomial number of rounds [85], which is too inefficient for use in practice. In any case, it may be advantageous to consider

variations of the model, where the passwords do not need to be extracted, but can be tested in some indirect manner (e.g., by asking the ring master whether the password is equal to the value of some hard-to-compute expression).

# Chapter 4

## General Conclusion

In summary, two cryptographic applications have been analyzed in this thesis: All-or-Nothing Transforms (AONTs), and Password-Authenticated Key Exchange protocols. For each of these, we have presented novel security definitions (in several flavors), gave efficient constructions, and shown the security of the constructions in our models. Our results on AONTs have already led to further research in that area. We hope that our work on password authentication will also have a substantial effect on future research, both in that specific area, as well as in the more general context of the construction of provably secure protocols. Our proof methods seem to be rather general, and may be of use for many schemes involving random oracles.

# Bibliography

- [1] L. M. Adleman, D. R. Estes, and K. S. McCurley. Solving bivariate quadratic congruences in random polynomial time. *Mathematics of Computation*, 48(177):17–28, Jan. 1987.
- [2] M. Ajtai and C. Dwork. A public-key cryptosystem with worst-case/average-case equivalence. In *Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing*, pages 284–293, El Paso, Texas, 4–6 May 1997.
- [3] D. Beaver. Secure multiparty protocols and zero-knowledge proof systems tolerating a faulty minority. *Journal of Cryptology*, 4(2):75–122, 1991.
- [4] M. Bellare and A. Boldyreva. The security of chaffing and winnowing. Available from IACR ePrint archive (report 2000/010), Jan. 2000. Preliminary draft.
- [5] M. Bellare, R. Canetti, and H. Krawczyk. A modular approach to the design and analysis of authentication and key exchange protocols. In STOC’98 [97], pages 419–428.
- [6] M. Bellare, A. Desai, D. Pointcheval, and P. Rogaway. Relations among notions of security for public-key encryption schemes. In CRYPTO’98 [31], pages 26–45.
- [7] M. Bellare, D. Pointcheval, and P. Rogaway. Authenticated key exchange secure against dictionary attacks. In EUROCRYPT2000 [40].
- [8] M. Bellare and P. Rogaway. Entity authentication and key distribution. In *Advances in Cryptology—CRYPTO ’93*, volume 773 of *Lecture Notes in Computer Science*, pages 232–249. Springer-Verlag, 22–26 Aug. 1993.

- [9] M. Bellare and P. Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *Proceedings of the First Annual Conference on Computer and Communications Security*. ACM, 1993. Latest version can be obtained from <http://www-cse.ucsd.edu/users/mihir/papers/ro.html>.
- [10] M. Bellare and P. Rogaway. Optimal asymmetric encryption. In *EUROCRYPT'94* [41], pages 92–111. Latest version can be obtained from <http://www-cse.ucsd.edu/users/mihir/papers/pke.html>.
- [11] M. Bellare and P. Rogaway. Provably secure session key distribution—the three party case. In *Proceedings of the Twenty-Seventh Annual ACM Symposium on the Theory of Computing*, pages 57–66, Las Vegas, Nevada, 29 May–1 June 1995.
- [12] M. Bellare and P. Rogaway. The exact security of digital signatures—how to sign with RSA and Rabin. In *Advances in Cryptology—EUROCRYPT 96*, volume 1070 of *Lecture Notes in Computer Science*, pages 399–416. Springer-Verlag, 12–16 May 1996.
- [13] S. M. Bellovin and M. Merritt. Encrypted key exchange: Password-based protocols secure against dictionary attacks. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, pages 72–84, 1992.
- [14] S. M. Bellovin and M. Merritt. Augmented encrypted key exchange: A password-based protocol secure against dictionary attacks and password file compromise. In *CCS'93* [25], pages 244–250.
- [15] M. Ben-Or, O. Goldreich, S. Micali, and R. L. Rivest. A fair protocol for signing contracts. *IEEE Transactions on Information Theory*, 36(1):40–46, 1990.
- [16] S. Blake-Wilson, D. Johnson, and A. Menezes. Key agreement protocols and their security analysis. In *Sixth IMA Intl. Conf. on Cryptography and Coding*, 1997.

- [17] M. Blaze. High-bandwidth encryption with low-bandwidth smartcards. In D. Grollman, editor, *Fast Software Encryption: Third International Workshop*, volume 1039 of *Lecture Notes in Computer Science*, pages 33–40, Cambridge, UK, 21–23 Feb. 1996. Springer-Verlag.
- [18] D. Boneh. The decision Diffie-Hellman problem. In *Proceedings of the Third Algorithmic Number Theory Symposium*, volume 1423 of *Lecture Notes in Computer Science*, pages 48–63. Springer-Verlag, 1998.
- [19] M. Boyarsky. Public-key cryptography and password protocols: The multi-user case. In CCS’99 [26], page to appear.
- [20] V. Boyko. On the security properties of OAEP as an all-or-nothing transform. In CRYPTO’99 [32].
- [21] V. Boyko, P. MacKenzie, and S. Patel. Provably-secure password authentication and key exchange using Diffie-Hellman. In EUROCRYPT2000 [40].
- [22] E. Brickell. The cryptanalysis of knapsack cryptosystems. In R. Ringeisen and F. Roberts, editors, *Applications of Discrete Mathematics*, pages 3–23. Society for Industrial and Applied Mathematics, 1988.
- [23] R. Canetti, Y. Dodis, S. Halevi, E. Kushilevitz, and A. Sahai. Exposure-resilient functions and all-or-nothing transforms. In EUROCRYPT2000 [40].
- [24] R. Canetti, O. Goldreich, and S. Halevi. The random oracle methodology, revisited. In STOC’98 [97], pages 209–218.
- [25] *Proceedings of the First Annual Conference on Computer and Communications Security*, 1993.
- [26] *Proceedings of the Sixth Annual Conference on Computer and Communications Security*, 1999.
- [27] B. Chor and R. L. Rivest. A knapsack type public-key cryptosystem based on arithmetic in finite fields (preliminary draft). In CRYPTO’84 [30], pages 54–65.

- [28] R. Cooper and W. Patterson. A generalization of the knapsack algorithm using Galois fields. *Cryptologia*, 8(4):343–347, 1984.
- [29] R. Cramer and V. Shoup. A practical public key cryptosystem provably secure against adaptive chosen ciphertext attack. In CRYPTO’98 [31], pages 13–25.
- [30] *Advances in Cryptology: Proceedings of CRYPTO 84*, volume 196 of *Lecture Notes in Computer Science*. Springer-Verlag, 1985, 19–22 Aug. 1984.
- [31] *Advances in Cryptology—CRYPTO ’98*, volume 1462 of *Lecture Notes in Computer Science*. Springer-Verlag, 17–21 Aug. 1998.
- [32] *Advances in Cryptology—CRYPTO ’99*, volume 1666 of *Lecture Notes in Computer Science*. Springer-Verlag, 15–19 Aug. 1999.
- [33] D. E. Denning and M. S. Sacco. Timestamps in key distribution protocols. *Communications of the ACM*, 24(7):533–536, Aug. 1981.
- [34] A. Desai. Indistinguishability of keys in symmetric encryption: Protecting against exhaustive key search, May 2000. Preliminary version. To appear in proceedings of CRYPTO 2000.
- [35] W. Diffie and M. Hellman. New directions in cryptography. *IEEE Trans. Info. Theory*, 22(6):644–654, 1976.
- [36] D. Dolev, C. Dwork, and M. Naor. Non-malleable cryptography (extended abstract). In *Proceedings of the Twenty Third Annual ACM Symposium on Theory of Computing*, pages 542–552, New Orleans, Louisiana, 6–8 May 1991.
- [37] C. Dwork, M. Naor, and A. Sahai. Concurrent zero-knowledge. In STOC’98 [97], pages 409–418.
- [38] T. ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithm. *IEEE Trans. Info. Theory*, 31:469–472, 1985.

- [39] D. Estes, L. M. Adleman, K. Kompella, K. S. McCurley, and G. L. Miller. Breaking the Ong-Schnorr-Shamir signature scheme for quadratic number fields. In *Advances in Cryptology—CRYPTO '85*, volume 218 of *Lecture Notes in Computer Science*, pages 3–13. Springer-Verlag, 1986, 18–22 Aug. 1985.
- [40] *Advances in Cryptology—EUROCRYPT '2000*, Lecture Notes in Computer Science. Springer-Verlag, 14–18 May 2000.
- [41] *Advances in Cryptology—EUROCRYPT 94*, volume 950 of *Lecture Notes in Computer Science*. Springer-Verlag, 1995, 9–12 May 1994.
- [42] S. Even, O. Goldreich, and A. Lempel. A randomized protocol for signing contracts. *Communications of the ACM*, 28:637–647, 1985.
- [43] H. Feistel. Cryptography and computer privacy. *Scientific American*, 228:15–23, May 1973.
- [44] E. Fujisaki and T. Okamoto. Secure integration of asymmetric and symmetric encryption schemes. In CRYPTO'99 [32], pages 537–554.
- [45] O. Goldreich. *Modern Cryptography, Probabilistic Proofs and Pseudorandomness*. Springer-Verlag, 1999.
- [46] S. Goldwasser and S. Micali. Probabilistic encryption. *J. Comput. Syst. Sci.*, 28(2):270–299, Apr. 1984.
- [47] S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof systems. *SIAM J. Comput.*, 18(1):186–208, Feb. 1989.
- [48] S. Goldwasser, S. Micali, and R. L. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM J. Comput.*, 17(2):281–308, Apr. 1988.
- [49] L. Gong. Optimal authentication protocols resistant to password guessing attacks. In *Proceedings of the 8th IEEE Computer Security Foundations Workshop*, pages 24–29, 1995.

- [50] L. Gong, T. M. A. Lomas, R. M. Needham, and J. H. Saltzer. Protecting poorly chosen secrets from guessing attacks. *IEEE Journal on Selected Areas in Communications*, 11(5):648–656, June 1993.
- [51] S. Halevi and H. Krawczyk. Public-key cryptography and password protocols. In *Proceedings of the Fifth Annual Conference on Computer and Communications Security*, pages 122–131, 1998.
- [52] N. Haller. The S/KEY one-time password system. *RFC 1760*, 1995.
- [53] J. Håstad, R. Impagliazzo, L. A. Levin, and M. Luby. A pseudorandom generator from any one-way function. *SIAM Journal on Computing*, 28(4):1364–1396, Aug. 1999.
- [54] R. Impagliazzo and M. Luby. One-way functions are essential for complexity based cryptography (extended abstract). In *30th Annual Symposium on Foundations of Computer Science*, pages 230–235, Research Triangle Park, North Carolina, 30 Oct.–1 Nov. 1989. IEEE.
- [55] R. Impagliazzo and S. Rudich. Limits on the provable consequences of one-way permutations. In *Proceedings of the Twenty First Annual ACM Symposium on Theory of Computing*, pages 44–61, Seattle, Washington, 15–17 May 1989.
- [56] D. Jablon. Strong password-only authenticated key exchange. *ACM Computer Communication Review, ACM SIGCOMM*, 26(5):5–20, 1996.
- [57] D. Jablon. Extended password key exchange protocols immune to dictionary attack. In *WETICE’97 Workshop on Enterprise Security*, 1997.
- [58] M. Jakobsson, J. Stern, and M. Yung. Scramble all, encrypt small. In *Fast Software Encryption: 6th International Workshop*, Rome, Italy, 24–26 Mar. 1999.
- [59] D. Johnson and S. Matyas. Asymmetric encryption: Evolution and enhancements. *CryptoBytes*, 2(1):1–6, Spring 1996. Available from <ftp://ftp.rsa.com/pub/crypto/bytes/crypto2n1.pdf>.

- [60] D. Johnson, S. Matyas, and M. Peyravian. Encryption of long blocks using a short-block encryption procedure. Available from <http://grouper.ieee.org/groups/1363/contributions/peyrav.ps>, Nov. 1996. Submitted for inclusion in the IEEE P1363a standard.
- [61] D. Kahn. *The Codebreakers*. Scribner, New York, revised edition, 1996.
- [62] P. Kocher, J. Jaffe, and B. Jun. Differential power analysis. In CRYPTO'99 [32], pages 388–397.
- [63] P. C. Kocher. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In *Advances in Cryptology—CRYPTO '96*, volume 1109 of *Lecture Notes in Computer Science*, pages 104–113. Springer-Verlag, 18–22 Aug. 1996.
- [64] L. Levin. One-way functions and pseudorandom generators. *Combinatorica*, 7:357–363, 1987.
- [65] T. M. A. Lomas, L. Gong, J. H. Saltzer, and R. M. Needham. Reducing risks from poorly chosen keys. *ACM Operating Systems Review*, 23(5):14–18, Dec. 1989. Proceedings of the 12th ACM Symposium on Operating System Principles.
- [66] S. Lucks. Open key exchange: How to defeat dictionary attacks without encrypting public keys. In *Proceedings of the Workshop on Security Protocols*, 1997.
- [67] P. MacKenzie and R. Swaminathan. Secure network authentication with password information. Manuscript.
- [68] S. Matyas, M. Peyravian, and A. Roginsky. Security analysis of Feistel ladder formatting procedure. Available from <http://grouper.ieee.org/groups/1363/contributions/peyrav2.ps>, Mar. 1997.

- [69] K. S. McCurley. The discrete logarithm problem. In C. Pomerance, editor, *Cryptology and Computational Number Theory*, volume 42 of *Proceedings of Symposia in applied Mathematics*. American Mathematical Society, 1990.
- [70] R. C. Merkle. Secure communications over insecure channels. *Communications of the ACM*, 21:294–299, Apr. 1978.
- [71] R. C. Merkle and M. E. Hellman. Hiding information and signatures in trapdoor knapsacks. *IEEE Transactions on Information Theory*, 24(5):525–530, Sept. 1978.
- [72] S. Micali, C. Rackoff, and R. H. Sloan. The notion of security for probabilistic cryptosystems. *SIAM J. Computing*, 17(2):412–426, Apr. 1988.
- [73] R. Motwani and P. Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995.
- [74] D. Naccache. Can O.S.S. be repaired? Proposal for a new practical signature scheme. In *Advances in Cryptology—EUROCRYPT 93*, volume 765 of *Lecture Notes in Computer Science*, pages 233–239. Springer-Verlag, 1994, 23–27 May 1993.
- [75] National Institute of Standards and Technology (NIST). *FIPS Publication 46: Announcing the Data Encryption Standard*, Jan. 1977. Originally issued by National Bureau of Standards.
- [76] National Institute of Standards and Technology (NIST). *FIPS Publication 180-1: Secure Hash Standard*, Apr. 1995.
- [77] R. M. Needham and M. D. Schroeder. Using encryption for authentication in large networks of computers. *Communications of the ACM*, 21(12):993–999, Dec. 1978.
- [78] R. M. Needham and M. D. Schroeder. Authentication Revisited. *Operating Systems Review*, 21(7):7–7, Jan. 1987.

- [79] P. Nguyen and J. Stern. Cryptanalysis of the ajtai-dwork cryptosystem. In CRYPTO'98 [31], pages 223–242.
- [80] H. Ong and C. P. Schnorr. Signatures through approximate representation by quadratic forms (extended abstract). In *Advances in Cryptology: Proceedings of Crypto 83*, pages 117–131. Plenum Press, New York and London, 1984, 22–24 Aug. 1983.
- [81] H. Ong, C. P. Schnorr, and A. Shamir. Efficient signature schemes based on polynomial equations (preliminary version). In CRYPTO'84 [30], pages 37–46.
- [82] S. Patel. Number theoretic attacks on secure password schemes. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, pages 236–247, 1997.
- [83] S. Patel, 1999. Personal communication.
- [84] J. Pollard and C.-P. Schnorr. Solution of  $x^2 + ky^2 \equiv m \pmod{n}$ , with application to digital signatures. *IEEE Transactions on Information Theory*, 22:702–709, 1987.
- [85] R. Richardson and J. Kilian. On the concurrent composition of zero-knowledge proofs. In *Advances in Cryptology—EUROCRYPT 99*, volume 1592 of *Lecture Notes in Computer Science*, pages 415–431. Springer-Verlag, 1999.
- [86] R. Rivest. All-or-nothing encryption and the package transform. In E. Biham, editor, *Fast Software Encryption: 4th International Workshop*, volume 1267 of *Lecture Notes in Computer Science*, pages 210–218, Haifa, Israel, 20–22 Jan. 1997. Springer-Verlag. Also available from <http://theory.lcs.mit.edu/~rivest/fusion.ps>.
- [87] R. Rivest. Chaffing and winnowing: Confidentiality without encryption. Available from <http://theory.lcs.mit.edu/~rivest/chaffing-980701.txt>, July 1998.

- [88] R. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signature and public key cryptosystems. *Communications of the ACM*, 21:120–126, 1978.
- [89] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.
- [90] M. Roe, B. Christianson, and D. Wheeler. Secure sessions from weak secrets. Technical report, University of Cambridge and University of Hertfordshire, 1998. Submitted to Operating Systems Review.
- [91] J. Rompel. One-way functions are necessary and sufficient for secure signatures. In *Proceedings of the Twenty Second Annual ACM Symposium on Theory of Computing*, pages 387–394, Baltimore, Maryland, 14–16 May 1990.
- [92] A. Shamir. A polynomial time algorithm for breaking the basic Merkle-Hellman cryptosystem (extended abstract). In *Advances in Cryptology: Proceedings of Crypto 82*, pages 279–288. Plenum Press, New York and London, 1983, 23–25 Aug. 1982.
- [93] V. Shoup. On formal models for secure key exchange. In CCS’99 [26], page to appear.
- [94] W. Simpson. PPP challenge handshake authentication protocol (CHAP). *RFC 1994*, 1996.
- [95] M. Steiner, G. Tsudik, and M. Waidner. Refinement and extension of encrypted key exchange. *ACM Operating System Review*, 29:22–30, 1995.
- [96] D. Stinson. Some observations on all-or-nothing transforms. Available from <http://cacr.math.uwaterloo.ca/~dstinson/papers/AON.ps>, 31 Aug. 1998.
- [97] *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing*, Dallas, Texas, 23–26 May 1998.

- [98] S. Vaudenay. Cryptanalysis of the Chor-Rivest cryptosystem. In CRYPTO'98 [31], pages 243–256.
- [99] G. S. Vernam. Cipher printing telegraph systems for secret wire and radio telegraphic communications. *J. Am. Inst. Elec. Eng.*, 55:109–115, 1926.
- [100] T. Wu. The secure remote password protocol. In *Proceedings of the 1998 Internet Society Network and Distributed System Security Symposium*, pages 97–111, 1998.
- [101] T. Wu. A real world analysis of Kerberos password security. In *Proceedings of the 1999 Internet Society Network and Distributed System Security Symposium*, 1999.