Computational Soundness of Formal Adversaries

by

Jonathan Herzog

Submitted to the Department of Electrical Engineering and Computer Science
on 29 Sept 2002, in partial fulfillment of the
requirements for the degree of
Master of Science

Abstract

The Dolev–Yao model is a useful and widespread framework in which to analyze security protocols. However, it models the messages of the protocol at a very high level and makes extremely strong assumptions about the power of the adversary. The computational model of cryptography and cryptographic protocols takes a much more low-level view of messages and uses much weaker assumptions. A major result of this work will be the demonstration that certain kinds of computational cryptography can result in an equivalence of sorts between the formal and computational adversary. Specifically, we give an interpretation to the messages of the Dolev–Yao model in terms of computational cryptography. We then define a computational security condition on the powers of the computational adversary, and show that this condition limits the computational adversary to the operations of the Dolev–Yao adversary. Lastly, we show that this security condition is achievable using standard computational cryptographic constructs.

Thesis Supervisor: Ron Rivest
Title: Professor, MIT

Thesis Supervisor: Nancy Lynch
Title: Professor, MIT

Computational Soundness of Formal Adversaries

by

Jonathan Herzog

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2002

Author . . . .
Department of Electrical Engineering and Computer Science
29 Sept 2002

Certified by . . . .
Ron Rivest
Professor, MIT
Thesis Supervisor

Certified by . . . .
Nancy Lynch
Professor, MIT
Thesis Supervisor

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

Contents

Chapter 1

Introduction

The area of formal cryptography has had two beginnings. The first came in 1978, when Needham and Schroeder proposed the first set of authentication protocols [14]. The second beginning came seventeen years later, when Gavin Lowe found a flaw in Needham and Schroeder's public key protocol, fixed the protocol, and — most importantly — proved that the fixed protocol was correct [9, 10]. What made these two beginnings different from previous cryptographic efforts was the level of abstraction. The protocols that Needham and Schroeder proposed were specified in terms of abstract cryptographic operations rather than specific cryptographic algorithms. Similarly, the flaw found by Lowe did not rely upon the properties of the cryptographic algorithms, and existed even in the abstracted system.

But what are authentication protocols? There is no exact definition, but the collection of examples shares many characteristics:

- The protocols are a sequence of messages between two or three parties,
- The messages utilize cryptographic algorithms to "secure" the contents in various ways,
- The protocol as a whole is intended to perform one or both of two objectives:
    1. Authentication: over the course of the protocol, one of the parties should gain proof that another particular party is also participating in the protocol, that they share common views of the protocol, and that they agree on the values used in the protocol, and
    2. Secrecy: that certain values used in the protocol should be unknown to other observers.

1.1 An Example Protocol

A particular example makes the definition much clearer. The Needham–Schroeder public key protocol can be described as a sequence of three messages between two parties:

1. $A \rightarrow B : \{\!|A\, N_a|\!\}_{K_B}$
2. $B \rightarrow A : \{\!|N_a\, N_b|\!\}_{K_A}$
3. $A \rightarrow B : \{\!|N_b|\!\}_{K_B}$

In this notation, A and B are names or unique identifiers of two parties. A starts the protocol, and hence has the role of *initiator*. B responds to A's original message, and hence has the role of *responder*.[1] The protocol begins with the step $A \rightarrow B : \{\!|A\, N_a|\!\}_{K_B}$. The notation $A \rightarrow B : M$, read "A sends to B the message M", is exactly that: the transmission of message M from entity A to entity B. The message of the first step is $\{\!|A\, N_a|\!\}_{K_B}$ to B, where

- $\{\!|M|\!\}_K$ is an encryption of the plaintext M with the key K, and
- $M_1\, M_2$ is the concatenation, or joining, of messages M_1 and M_2.

The value N_a is a *nonce*, a random value generated freshly by (in this case) A, and of sufficient length to make infinitesimally small the chances of its previous use by any other party.[2] N_b is a nonce generated by B, which it includes in the second message.

This protocol has two goals:

1. Secrecy: The two values N_a and N_b should be known only to A and B. More generally, the nonces used in the protocol should only be known to the two participants, and
2. Authentication: The initiator and responder should be authenticated to each other. Specifically:
   - The initiator should know the identity of the responder,[3] that the responder knows who the initiator is, and that they agree on the values of the nonces used, and
   - The responder should know the identity of the initiator, that the initiator knows who the responder is, and that they agree on the values of the nonces used.

[1]Though it is conceivable to have a protocol without analogous roles, they are always found in practice. Also, it is not unusual for the initiator and responder to use the services of a trusted third party called the *server*, which is often found in other protocols.

[2]In much of the literature, this is the definition of "nonce". There are other, weaker, definitions of nonce which require only that the value of the nonce be fresh, but many protocols rely upon the fact that the value of the nonce is unpredictable.

[3]That is, the initiator should know that the responder knows the appropriate secret key.

However, the authentication conditions do not hold. In the attack that Lowe discovered, the initiator (A) starts a run of the protocol with a malicious entity (M), who uses it to pretend to be the initiator ($M(A)$) to a third party (B):

1. $A \rightarrow M : \{\!|A, N_a|\!\}_{K_M}$
2. $M(A) \rightarrow B : \{\!|A, N_a|\!\}_{K_B}$
3. $B \rightarrow A : \{\!|N_a, N_b|\!\}_{K_A}$
4. $A \rightarrow M : \{\!|N_b|\!\}_{K_M}$
5. $M(A) \rightarrow B : \{\!|N_b|\!\}_{K_B}$

The entity B is fooled by the above attack. From B's perspective, the sequence of messages is exactly what it would expect from a run with the initiator A. But while the initiator A is engaged in a run of the protocol, it thinks that the responder is M, not B.

Lowe's fix is to include the responder's name in the second message, making it:

2. $B \rightarrow A : \{\!|B\, N_a\, N_b|\!\}_{K_A}$

The proof of correctness involved two parts: he first proved that any attack on a large system (i.e., multiple honest parties) could be translated into an attack on the small system of just two parties. He then used a model checker, a standard tool in the formal methods community, to exhaustively search the small system for vulnerabilities.

Since Lowe's first two papers on the subject, there has been a flurry of interest in the area. The use of model checkers has been expanded and refined [13], theorem provers have been shown to be of value [15], and more direct mathematical methods have proved to be quite useful [17]. However, all these methods share common characteristics, and hence common weaknesses. All of these methods use the Dolev–Yao model [6] to model the attacker, and some aspects of this model have no theoretical foundation.

1.2 The Dolev–Yao Model

The Dolev–Yao model is an early and successful effort to provide a mathematical framework in which these protocols can be examined. In this model, there are two kinds of active parties: *regular* (or *honest*) participants and the adversary. The regular participants follow the steps of the protocol without deviation. They can engage in multiple runs of the protocol simultaneously, with different parties. The model contains only the messages they send and receive; internal states are not modeled explicitly. The messages they send and receive are assumed to be sent over a shared network, and messages are tagged with (unsecured) source and destination values.

The messages are assumed to be elements of an algebra $\mathcal{A}$ of values. There are two types of atomic messages, texts ($\mathcal{T}$) and keys ($\mathcal{K}$). Compound messages are created by two deterministic operations:

- $\mathit{encrypt} : \mathcal{K} \times \mathcal{A} \rightarrow \mathcal{A}$
- $\mathit{pair} : \mathcal{A} \times \mathcal{A} \rightarrow \mathcal{A}$

We write $\{\!|M|\!\}_K$ for $\mathit{enc}(K, M)$ and $M\,N$ for $\mathit{pair}(M, N)$.[4] We also write $\mathcal{E}$ ("encryptions") for the range of *encrypt*, and $\mathcal{C}$ ("pairs") for the range of *pair*. Additionally, the algebra is assumed to be *free*; every value can be produced by applying the above operators to atomic elements in a unique way. Phrased differently, every value has a unique representation.

The network is assumed to be completely under the control of the adversary, who can record, delete, replay, reroute, and reorder messages. The adversary is also able to create new messages, to a limited degree. In this setting, the adversary is not usually thought of as having a particular goal. Instead, the adversary is thought of as a source of noise; a non-deterministic process that can send any message it can generate to any participant at any time. The main approach to proving security in this model is to find the strongest security properties (secrecy and/or authentication) achievable by the regular participants despite the presence of the adversary.

The ability of the adversary to create new messages is limited:

- The adversary is assumed to know all public or predictable values, such names, public keys and time-stamps.
- It can also create fresh values, such as nonces, session keys, and names.
- It is assumed to have at least one name, and it is possible that regular participants may communicate with the adversary as if it were another regular participant. It can create new names for itself on the fly.
- It is assumed to know some set of secret or private keys. These keys would include those which it would know as a regular participant. If the adversary is a made up of multiple entities[5] this set would include the secret keys of each entity.

[4]When three or more terms are written together, such as $M_1\,M_2\,M_3$, we assume they are grouped to the left. That is, $M_1\,M_2\,M_3 = \mathit{pair}(\mathit{pair}(M_1, M_2), M_3)$.

[5]As it would be if it were a coalition of adversaries, or if the adversary has corrupted some honest participants. In these cases, however, we assume the coalition remains constant and that the adversary does not corrupt any new entities during a protocol run. To model the corruption of a participant, it is usually assumed that the participant was corrupted before the protocol run began.

- It can encrypt values it knows with keys it knows,
- It can decrypt encryptions, if it knows the decryption key,[6] and
- It can pair two known values into a compound message, or separate the two parts of a pair.

Other than these explicitly enumerated abilities, the adversary has no powers.

Two assumptions in the model — the freeness of the algebra and the limitations on the adversary — are surprisingly strong:

- The freeness assumption simply does not hold for many encryption schemes in widespread use, such as DES-CBC and Rijndael-CBC[7]. In these schemes, the number of possible ciphertexts is much less than the number of plaintext/key pairs (for a fixed message length). The inevitability of ciphertext collisions is not necessarily worrisome: the freeness assumption would still seem to be true in an approximate sense for these schemes. However, it is a very bad assumption to make for encryption schemes such as one-time pads, which are secure in an information-theoretic sense. There is also no *a priori* reason that freeness need be true for any arbitrary encoding and encryption scheme.
- The limitations on the adversary are an even stronger set of assumptions. For example, in the basic RSA encryption algorithm, the plaintext M and the ciphertext $\{\!|M|\!\}_k$ are represented as integers. Due to the internal workings of RSA, it turns out that $\{\!|M_1|\!\}_k \cdot \{\!|M_2|\!\}_k = \{\!|M_1 \cdot M_2|\!\}_k$. Hence, when RSA is used the adversary has an additional operation: transforming the encryption of two plaintext into the encryption of the plaintexts' product.

  Even for an arbitrary encryption scheme, it is very possible that the adversary could have additional operations such as being able to change the plaintext of an encryption even without knowing what the plaintext is, or being able to create a valid encryption without knowing what is encrypted. These additional operations are usually undesirable, and considerable research has gone into identifying and eliminating these additional possibilities. The cryptography used in practice, however, lags behind the state of the research and often contains such undesirable possibilities.

These two assumptions — the freeness of the algebra and the limitations on the adversary — can be relaxed somewhat, but not entirely disregarded. Variants of the Dolev–Yao model which contain

[6]Note that these operations are distinct from the operations used to define the algebra $\mathcal{A}$. Even though the adversary can decrypt, for example, there is no decryption operator used in the construction of $\mathcal{A}$.

[7]That is, DES or Rijndael in cipher-block chaining mode.

additional and/or abstract operations have been studied [15], and some attempts at non-freeness have been attempted [8].

However, while the assumptions listed above may be stronger than necessary, assumptions like them must be present in any model inspired by or derived from Dolev–Yao. All the methods that work in this model operate in a recursive manner: they first examine all the possible operations available to the adversary that could produce a “bad” term. Then, they consider all the inputs those operations would require, and consider all operations available to the adversary that would produce any of *those* terms. The process is repeated indefinitely, until the entire set of “dangerous” terms is defined. If none of these dangerous terms are available to the adversary — either initially, or as part of a protocol run — then the original “bad” term cannot be produced.

It is essential to this style of argument that the set of possible operations which could produce a “bad” term be explicitly enumerated. The recursive process will fail to catch all “dangerous” terms if the set of possible operations is incomplete. Also, for tractability, it is necessary that the number of ways to produce any given term be small. The more ways there are to produce a given term, the greater the number of “attacker strategies” that must be checked as part of a correctness proof (increasing the resources required by automatic tools such as model checkers).

Hence, the strong assumptions of the Dolev–Yao model. By stipulating that the algebra be free and the adversary limited, the model restricts the possible ways in which a term could be constructed to a small number. True, the adversary can produce a term M by decrypting $\{\!|M|\!\}_K$, but that requires the adversary to possess both $\{\!|M|\!\}_K$ and K^{-1}. The recursively-defined set of dangerous terms ends up being fairly small. But if a given term, such as g^{xy} or $Y \oplus X_r$, could be produced in many different ways because of the underlying algebra, or if the adversary has an unbounded number of operations, then almost every term can be considered dangerous and these methods fail.

Hence, to use the methods that build on the Dolev–Yao model, one must first make assumptions similar to those above. It is not clear, however, that these assumptions are at all justified, and they cast doubt onto any method that uses them. It is true that many attacks and flaws have been found even in the presence of these assumptions. However, if a method based on these assumptions finds no flaws in a protocol, there still remains the possibility that some attack can be found when the adversary has an additional ability.

1.3 The Computational Model

The assumptions of the Dolev–Yao model seem especially strong when compared to those of the widely-accepted model of encryption and cryptographic algorithms: that of "computational" cryptography. This approach regards cryptography as a branch of complexity theory, and regards cryptographic primitives as algorithms that map bit-strings into bit-strings. Adversaries are regarded as probabilistic polynomial-time (PPT) algorithms, and security conditions are stated in terms of the probability that an adversary can perform a given calculation in the face of an increasing "security parameter."

For example, suppose that $\{D_\eta\}$ and $\{D'_\eta\}$ are two families of distributions on $\{0,1\}^*$, indexed by the security parameter $\eta \in \mathsf{N}$. Then the two distributions are said to be *computationally indistinguishable* if all PPT algorithms have only a negligible chance of distinguishing a sample from D_η from D'_η. More formally, let $\mathtt{T}_{PPT} : \{0,1\}^* \rightarrow \{0,1\}$ be a *distinguisher*. Then we can define computational indistinguishability in the following way:

Definition 1 *Two distribution families, $\{D_\eta\}$ and $\{D'_\eta\}$ are* computationally indistinguishable *(written $\{D_\eta\} \cong \{D'_\eta\}$) if*

$$\forall\, \mathtt{T}_{PPT}\ \forall \textit{ polynomials } q, \exists\, \eta_0\ \forall\, \eta > \eta_0$$
$$\left|\Pr\left[x \leftarrow D_\eta : \mathtt{T}(x) = 0\right] - \Pr\left[x \leftarrow D'_\eta : \mathtt{T}(x) = 0\right]\right| < \frac{1}{q(\eta)}$$

(Here, $x \leftarrow D$ means that x is drawn from distribution D, and $\Pr[a : p]$ is an alternative notation for $\Pr[p|a]$.)

In the computational framework, computational indistinguishability is used to express many different types of security conditions. In the case of pseudo-random generators, for example, a generator is pseudo-random if the output of the generator, when given a random seed, is computationally indistinguishable from a random string of the same length. In the case of encryption, the security condition would be that no matter how two messages are chosen their encryptions (under a random key) would be indistinguishable. (Here, the security parameter would be the length of the key, while in the previous example the security parameter would be the length of the random seed.)

In practice, one proves that any given algorithm meets the relevant definition of security by proving that if it did not, one could solve some underlying "hard" problem. For most cryptographic primitives, the notion of a probabilistic polynomial time adversary is sufficient. For authentication protocols, on the other hand, a more advanced adversary is needed.

The "oracle model" [2] models authentication protocols in the computational framework. Messages are represented as bit-strings, and encryption operations are assumed to meet certain standard computational definitions of security such as the two suggested above. The main difference is how the regular participants are modeled. Here, regular participants are oracles which will transform well-formed bit-strings into well-formed bit-strings. The adversary, then, is modeled as an arbitrary probabilistic, polynomial-time algorithm which now has oracle access to regular participants. Aside from its complexity, no limitation is placed on the adversary: it can create arbitrary bit strings or perform any polynomial time calculation. Security is often expressed in two ways:

- Authentication is modeled as a predicate on oracles. Usually, the predicate states that for every oracle queried by the adversary, there is another oracle queried by the adversary in a "matching" way. (That is, if a regular participant of the first oracle observes a sequence of queries, then the regular participant of the second oracle must observe a particular, related sequence.)
- Secrecy is modeled by indistinguishability. That is, the secrecy condition holds if no adversary can distinguish between the secret value/key and a random value of the same length.

As is characteristic of the computational framework, protocols are almost always proven secure by a reduction argument. For example, some protocols are designed to use number theoretic problems directly. In this case, one would typically show that if there exists a probabilistic, polynomial-time algorithm $\mathtt{A}$ which is able to violate either of the above conditions, then there exists another probabilistic polynomial-time algorithm $\mathtt{A}'$ which can solve a "hard" problem. Alternately, a protocol might make use of abstract, lower-level cryptographic primitives (such as encryption) which have their own security properties. In this case, one would show that if A can violate the security conditions of the protocol, there exists another algorithm A' that violates the security conditions of the underlying cryptographic building blocks.

This model also has its advantages and disadvantages. Since the model uses a much lower level of abstraction, it is easier to have faith in the guarantees a proof would provide. However, the proofs in this model tend to be somewhat difficult. Proofs need to be created for each protocol individually; it is difficult to prove theorems about protocols in general.

1.4 Present Work

Given how these two models complement each other, it seems an enticing goal to unify them in some way to gain the benefits of both. While the oracle model uses its low level of abstraction to gain strong proofs, the Dolev–Yao model leverages its high level of abstraction to provide intuitive, simple, and reusable proof methods. If the assumptions of the Dolev–Yao model could be justified in some computational setting, then protocol designers could use high-level methods to yield low-level proofs.

Some work in this area has already been performed, particularly that of Abadi and Rogaway [1]. In that paper, the authors give significant support to the validity of the Dolev–Yao model by grounding a large part of the model in terms of computational cryptography. In particular, they identify the conditions secret-key cryptography must satisfy to insure that terms "indistinguishable" to the Dolev–Yao adversary are in fact "indistinguishable" to a computational adversary.

However, the results of that paper are only valid against a rather weak adversary which we will call a *passive distinguisher.* A *passive* adversary is one with no access to information other than its input. The name denotes the fact that while it can overhear the messages and activity of honest parties, it can send no messages of its own creation nor take any action to affect the honest parties. Likewise, a *distinguisher* is an adversary with the relatively difficult task of distinguishing two messages. Hence, the adversary of Abadi and Rogaway is itself weak, but charged with solving a difficult task.

The adversary that we would like to consider in this work is an *active synthesizer.* An *active* adversary is one that can create new messages and send them to honest participants in an attempt to solicit additional and potentially useful information. Calling an adversary a *synthesizer*[8] means that its goal is that of creating a new message. This is an easier task than distinguishing between two messages: the flaw of RSA described in Chapter 1.2 indicates that one can create $\{\!|M_1M_2|\!\}_K$ from $\{\!|M_1|\!\}_K$ and $\{\!|M_2|\!\}_K$, even if both $\{\!|M_1|\!\}_K$ and $\{\!|M_2|\!\}_K$ are indistinguishable from any other ciphertext. Hence, such an adversary would be relatively strong, and charged with accomplishing a relatively simple task. In this work, however, we are concerned with a slightly more difficult challenge. It will not be enough for the synthesizer to create any message, but only certain ones: messages which not could be created by the formal adversary. Even though this is more difficult than the creation of any single message, it is still a much easier task than that faced by the distinguisher.

The main results of this paper are three-fold:

1. We define a security property for computational cryptography called *ideal* security (Definition 24).

2. Abadi and Rogaway devised a method of relating the abstract messages of the formal model with computational "messages" (i.e., probability distributions over bit-strings) which uses computational cryptography. We show that if the primitives in this relation are ideal, then

[8]A term of my own creation. There may already be a standard term for this kind of adversary.

there is an equivalence of sorts between the active synthesizer and the Dolev-Yao adversary. Put another way, if the underlying cryptography is ideal, then no active synthesizer can perform an action unachievable by the Dolev-Yao adversary (Theorem 25).

3. We provide an example of an encryption algorithm that is ideal, and prove that it is so. (Theorem 26).

We review the Dolev–Yao and computational models in more depth (Chapters 2 and 3 respectively.) We explore how the messages of these two models relate in Chapter 4. We develop the adversary model relevant to this setting in Chapter 5. In Chapter 6 we consider ideal cryptography, and how it induces an equivalence between the active synthesizer and the formal adversary. We provide an example of an ideal encryption scheme in Chapter 7 along with a proof of its ideality. We will finish with some possible extensions and open problems in Chapter 8.

Chapter 2

Formal Preliminaries

As stated before, the messages in this model are assumed to be elements of an algebra $\mathcal{A}$ of values. The types of atomic messages will differ from application to application; in this setting we will be considering the case of asymmetric encryption, and using the Needham–Schroeder protocol as a running example. Hence, we will use two types of atomic messages, each having two subtypes:

- Texts ($\mathcal{T}$) with the two subtypes
  - names (public, predictable) ($\mathcal{M}$),
  - nonces (private, random, unpredictable) ($\mathcal{R}$),
- Keys ($\mathcal{K}$) with the two subtypes
  - public keys ($\mathcal{K}_{Pub}$), and
  - private keys ($\mathcal{K}_{Priv}$)

Compound messages are created by two deterministic operations:

- encrypt : $\mathcal{K}_{Pub} \times \mathcal{A} \to \mathcal{A}$ (Range: $\mathcal{E}$)
- pair : $\mathcal{A} \times \mathcal{A} \to \mathcal{A}$ (Range: $\mathcal{C}$)

Simple terms are those that are not pairs: atomic terms and encryptions. Additionally, the algebra is assumed to be *free*; every value can be produced by applying the above operators to atomic elements in a unique way. We require that there be a bijection

$$inv : \mathcal{K}_{Pub} \to \mathcal{K}_{Priv}$$

and we write $inv(K) = K^{-1}$. We also assume there is a one-one mapping

$$keyof : \mathcal{M} \to \mathcal{K}_{Pub}$$

where we write $keyof(A)$ as K_A.

Very often, the security of a protocol hinges on the fact that a certain value could not have been created by the adversary, despite the fact that the adversary has the power to create fresh random values. In particular, the adversary is assumed to have the power to create fresh random nonces and keys. Although we will discuss this issue in greater detail in Chapter 4, we should now define the set $\mathcal{R}_{Adv} \subseteq \mathcal{R}$ of nonces containing all the nonces which can be created by the formal adversary. There is nothing about these values which distinguishes them as elements of $\mathcal{R}_{Adv}$ versus nonces in $\mathcal{R} \setminus \mathcal{R}_{Adv}$, but this distinction will be useful later. Likewise, we will also define the set $\mathcal{K}_{Adv} \subseteq \mathcal{K}$ of keys which can be created by the adversary. We assume that the adversary has subverted some static set $Subv \subseteq \mathcal{M}$ of principals, and hence has access to their keys

$$\mathcal{K}_{Subv} = \left\{K_A^{-1} : A \in Subv\right\}.$$

Principals in *Subv* have been subverted by the adversary after their keys have been chosen. As opposed to the keys in $\mathcal{K}_{Adv}$, the adversary is unable to choose the values of the keys in $\mathcal{K}_{Subv}$ itself. We designate the entire set of private keys known to the adversary as $\mathcal{K}_{Bad} = \mathcal{K}_{Adv} \cup \mathcal{K}_{Subv}$.

In the formal setting, the global system is modeled as a set of communicating processes. That is, honest participants and the adversary are both modeled as processes that maintain internal state, and can send and receive messages on some channel. There are several different models in this style, each making slightly different assumptions about timing, synchronicity of the channels, receptiveness of the processes to messages, and so on. However, the essential aspects of the modeling are constant:

1. First, the adversary is assumed to have total control over the network. The adversary can repeat, delete, delay, and re-route messages as it sees fit. This is modeled by letting the adversary *be* the network. That is, in the model, the honest participants only send their messages to the adversary and only receive messages from the adversary (illustrated in Figure 2-1).

2. The model does not concern itself with any liveness properties. That is, the purpose of the model is to prove that "bad things" won't happen, not that "good things" will. For example, an adversary as powerful as that of the previous paragraph will be able to prevent any message from every reaching any honest participant. Thus, it is impossible to show that any honest

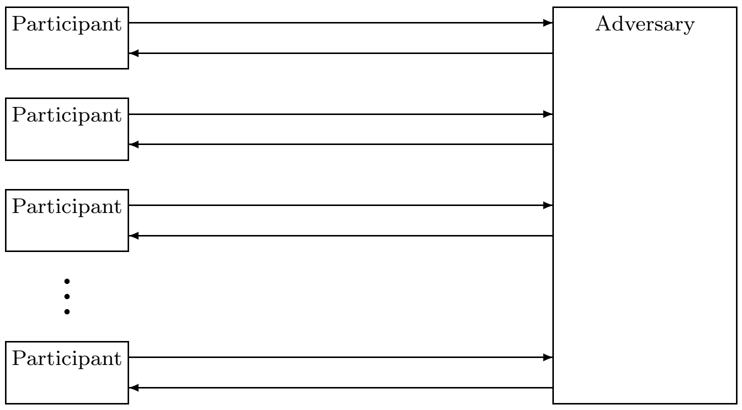

Figure 2-1: General formal model of communications

participant will ever complete a run of the protocol. From our perspective, this is perfectly acceptable: if no honest participant finishes the protocol, then no honest participant can ever be fooled using the protocol.

3. The model generally assumes that every message that reaches an honest participant is one that the participant is "expecting" to hear as part of the protocol. This is slightly different than issues of receptiveness[1]. The assumption is that the honest participants will never need to deal with a message that is ill-formed, or invalid for the participant's current stage of the protocol. This is certainly an assumption that we would like to address in the future, but our purpose here is to give a computational soundness to the model as it is. Strengthening the model can come afterward.

Given these aspects, and the fact that in this work we are interested almost entirely in the behavior of the adversary, we can abstract the model to a more convenient form (Figure 2-2). We will abstract the actions and capabilities of all honest participants into an "environment" process, with which the adversary communicates. That is, since the adversary has total control over the network, every communication is between the adversary and some honest participant. Hence, from the perspective of the adversary, each regular participant can be thought of as an interface to the environment. True, by going from many channels to one, we lose all addressing information. However, that information is insecure and under the control of the adversary, and we are not concerned with liveness properties. We simply regard the environment as non-deterministically routing each message from the adversary to a receptive participant.

[1]That is, whether or not the participant is able to receive a message at a given point in time.

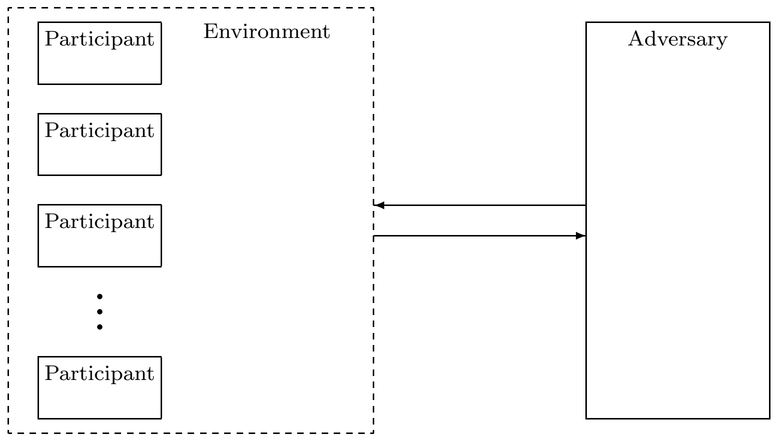

Figure 2-2: Our model of communication

One other commonality between formal models is the operation of the adversary. In addition to the power over the network, the adversary is assumed to have limited power to synthesize new messages to transmit. In particular, the adversary is assumed to have an internal knowledge set which can increase over time. The initial knowledge set of the adversary is usually assumed to consist of five things:

1. the public keys ($\mathcal{K}_{Pub}$),
2. the private keys of subverted participants ($\mathcal{K}_{Subv}$),
3. the names of the principals ($\mathcal{M}$),
4. the nonces it itself generates ($\mathcal{R}_{Adv}$) which are assumed to be distinct from all nonces generated by honest participants, and
5. the keys it itself generates ($\mathcal{K}_{Adv}$).

So, any analysis of the protocol must assume the adversary has access to at least these five sets of terms. If the adversary has learned any other set S of messages, then in sum total it knows the union of S with these five sets. For ease of notation, we will denote this union as:

Definition 2 *Let $S \subseteq \mathcal{A}$. Then the* knowledge set over S *is* $\widehat{S} = S \cup \mathcal{M} \cup \mathcal{K}_{Pub} \cup \mathcal{K}_{Subv} \cup \mathcal{K}_{Adv} \cup \mathcal{R}_{Adv}$.

Now, if the adversary knows the set $\widehat{S}$, then it can close it under the following operations:

- decryption with private keys in the knowledge set,
- encryption with public keys in the knowledge set,

- Deduce from an encryption the public key used to encrypt,
- joining of two elements in the knowledge set, and
- separation of a "join" element into its component elements.

To put it more formally:

Definition 3 *The* closure of $\widehat{S}$*, written* $C\left[\widehat{S}\right]$*, is the smallest set such that:*

1. $\widehat{S} \subseteq C\left[\widehat{S}\right]$,
2. *If* $\{\!|M|\!\}_K \in C\left[\widehat{S}\right]$ *and* $K^{-1} \in C\left[\widehat{S}\right]$*, then* $M \in C\left[\widehat{S}\right]$,
3. *If* $M \in C\left[\widehat{S}\right]$ *and* $K \in C\left[\widehat{S}\right]$*, then* $\{\!|M|\!\}_K \in C\left[\widehat{S}\right]$,
4. *If* $M\,N \in C\left[\widehat{S}\right]$*, then* $M \in C\left[\widehat{S}\right]$ *and* $N \in C\left[\widehat{S}\right]$,
5. *If* $M \in C\left[\widehat{S}\right]$ *and* $N \in C\left[\widehat{S}\right]$*, then* $M\,N \in C\left[\widehat{S}\right]$*, and*
6. *If* $\{\!|M|\!\}_K \in C\left[\widehat{S}\right]$*, then* $K \in C\left[\widehat{S}\right]$.

If, at any given point in time, the adversary has the knowledge set $\widehat{S}$, then it can send any element $M \in C\left[\widehat{S}\right]$ to any honest participant. If the adversary receives a response N from an honest participant, it replaces its knowledge set $\widehat{S}$ with $\widehat{S} \cup \{N\}$. It then can send any element of $C\left[\widehat{S} \cup \{N\}\right]$ to any participant, and so on. It is the central assumption of the formal model that this is the extent of the adversary's ability to manipulate cryptographic material. (Because we are concerned only with the closure of $\widehat{S}$ rather than of an arbitrary S, the last condition of the definition of closure is usually irrelevant.)

So, if the adversary is defined by this closure operation, then we can define an attack of the adversary against environment *Env* to be a possible trace of such an adversary when communicating with *Env*. That is, an attack will be a sequence of messages sent and received by the adversary, with certain restrictions. In general, however, there is no restriction on how many messages the adversary will receive between transmissions, or how many it will send between receptions. Since the order of message reception does not affect the operation of the adversary, we can model the trace as a sequence of message transmissions, alternating with reception of (possibly empty) message sets originating from *Env*:

Definition 4 *A* formal attack against environment *Env is a trace of a formal adversary in communication with the environment Env. That is, a sequence of message transmissions* ($q_i \in \mathcal{A}$) *and* *(possibly empty) message set receptions ($r_i \subseteq \mathcal{A}$):*

$$r_0 \quad q_1 \quad r_1 \quad q_2 \quad r_2 \quad \ldots \quad q_{n-1} \quad r_{n-1} \quad q_n \quad r_n$$

such that

$$q_i \in C\left[\widehat{\bigcup_{j=0}^{i-i} r_j}\right].$$

(The requirements on the sets r_i are determined by the specification of the environment Env.)

As stated above, the Dolev–Yao adversary is not assumed to have any particular goals. Instead, one attempts to prove the strongest security conditions that are still true despite the presence of such an adversary.

Chapter 3

Computational Preliminaries

On the computational side, the adversary is a probabilistic, polynomial-time Turing machine which is assumed to be attempting to violate some security property. Messages are finite-length bit-strings. Public-key encryption is not a simple operation, but a triple of algorithms $(\mathsf{G}, \mathsf{E}, \mathsf{D})$:

- $\mathsf{G} : \mathsf{Parameter} \times \mathsf{Coins} \rightarrow \mathsf{PublicKey} \times \mathsf{PrivateKey}$ is the (randomized) key generation algorithm,
- $\mathsf{E} : \mathsf{String} \times \mathsf{Coins} \times \mathsf{PublicKey} \rightarrow \mathsf{Ciphertext}$ is the (randomized) encryption algorithm, and
- $\mathsf{D} : \mathsf{String} \times \mathsf{PrivateKey} \rightarrow \mathsf{String} \cup \{\perp\}$ is the decryption algorithm, which we assume returns $\perp$ whenever the input string is not a valid encryption under the corresponding public key.

The helper sets are:

- $\mathsf{Parameter} = \mathsf{N}$
- $\mathsf{Coins} = \{0,1\}^{\omega}$, the set of all infinite sequences of bit-strings.
- $\mathsf{PublicKey}$, $\mathsf{PrivateKey}$ and $\mathsf{Ciphertext}$ vary between key generation algorithms and implicitly depend on the parameter, and
- $\mathsf{String} = \{0,1\}^*$

We will write $\mathsf{E}_k(m)$ for the probability distribution induced by $\mathsf{E}(m, r, k)$ where r is chosen randomly (uniformly) from Coins.[1] Similarly, we will write $\mathsf{G}(1^\eta)$ for the probability distribution induced by

[1]Technically, the distribution of r is somewhat more complex, but achieves the same intuitive result. Since the algorithms of an encryption scheme are assumed to take time polynomial in the security parameter (or the length of the keys, which are themselves polynomial in the security parameter) each algorithm can only depend on some finite sub-sequence of the random coins. Without loss of generality, we can assume the algorithms only depends on the first $q(\eta)$ bits, where q is some polynomial. Thus, Coins can be partitioned into equivalence classes, where all strings

$\mathsf{G}(1^\eta, r)$ where r is chosen randomly (uniformly) from Coins. We will also write $x \leftarrow \mathcal{D}$ to mean that x is drawn from the distribution $\mathcal{D}$. If $\mathcal{D}$ is a finite set, then we mean the uniform distribution over that set.

The weakest form of security for a public-key encryption algorithm is known as *semantic security* [12], and is best described as a game. First, a key-pair is generated according to the algorithm G. Then, the adversary (first represented by $\mathtt{M}$) picks two messages. One of those two messages, chosen at random, in encrypted. The adversary (now represented by $\mathtt{A}$) must then decide which of the two messages was encrypted. The encryption scheme is semantically secure if no adversary can make a correct guess non-negligibly greater than $\frac{1}{2}$.

Definition 5 *A public-key encryption algorithm* $(\mathsf{G}, \mathsf{E}, \mathsf{D})$ *is* one-pass semantically secure[2] *if:*

$$\forall \textit{ PPT algorithms } \mathtt{M} \textit{ and } \mathtt{A}, \forall \textit{ polynomials } q, \forall \textit{ sufficiently large } \eta,$$

$$\begin{aligned} \Pr[\quad & (e,d) \leftarrow \mathsf{G}(1^\eta); \\ & m_0, m_1 \leftarrow \mathtt{M}(1^\eta, e); \\ & b \leftarrow \{0,1\}; \\ & c \leftarrow \mathsf{E}_e(m_b) : \\ & \mathtt{A}(1^\eta, e, m_0, m_1, c) = b \quad] \leq \tfrac{1}{2} + \tfrac{1}{q(\eta)} \end{aligned}$$

The definition above is slightly different from the better-known form of security, the *three-pass* version [12]. The only difference is that in the three-pass version of this definition, the adversary $\mathtt{M}$ is not allowed to know the encryption key e when choosing the messages m_0 and m_1.[3] The one-pass version of the definition is strictly stronger than the three-pass version, and we will need the additional strength in our proofs.

We will later use another, apparently new, definition for public-key cryptography: that where one can extract the public key used for an encryption from the encryption itself:

Definition 6 *A public-key encryption scheme* $(\mathsf{G}, \mathsf{E}, \mathsf{D})$ *is* key-providing *if there exists a polynomial-*

with the first $q(\eta)$ bits are in the same class. Note that for any η, there are only a finite number of such equivalence classes. Hence, to select "randomly" from Coins, we first select an equivalence class, then select an arbitrary r from that class.

This is the intended interpretation of "selecting randomly from Coins that we will use in this paper. Likewise, when we speak of selecting a specific element of Coins, we mean setting the first $q(\eta)$ bits.

[2]Technically, the definition given here is that for a different definition of security, usually called *general message (GM) security*. However, GM-security is well-known to be equivalent to semantic security, and it has a slightly more convenient form for our purposes.

[3]The relevant line would be

$$m_0, m_1 \leftarrow \mathtt{M}(1^\eta)$$

in the three-pass version.

time algorithm Q *such that for all messages* m

$$\Pr[(e,d) \leftarrow \mathsf{G}(1^\eta);\ c \leftarrow \mathsf{E}_e(m) : \mathsf{Q}(c, 1^\eta) = e] = 1$$

Constructing a key-providing encryption scheme is no more difficult than constructing an encryption scheme:

Theorem 7 *If a semantically secure public-key encryption scheme exists, a semantically secure key-providing public-key encryption scheme exists.*

Proof: If $(\mathsf{G}, \mathsf{E}, \mathsf{D})$ is a semantically secure encryption scheme, then let

- $\mathsf{G}' = \mathsf{G}$
- E' be the algorithm that on input $\langle e, m\rangle$, runs $\mathsf{E}_e(m)$ to produce c, and then outputs $\langle c, e\rangle$, and
- $\mathsf{D}'(x) = \mathsf{D}(x[1])$, i.e. D on the first component of x.

Clearly, the scheme $(\mathsf{G}', \mathsf{E}', \mathsf{D}')$ is key-providing, since the public key is part of the ciphertext. Furthermore, if the original encryption scheme is semantically secure, then the modified encryption scheme will be as well. (Since the adversary $\mathtt{A}$ receives the public key e anyway, having the ciphertext reveal e gives $\mathtt{A}$ no additional power.) ■

Later in this work, we will make use of a particularly useful creature known as the *zero-knowledge (ZK) proof* [7]. A ZK proof is a type of interactive proof system, where an interactive proof is a sequence of messages by which a prover $\mathtt{P}$ convinces a verifier $\mathtt{V}$ of the truth of a theorem $x \in L$ (for some $\mathcal{NP}$ language L). In this work, we will concern ourselves with a special type of interactive proof called *non-interactive* zero-knowledge (NIZK) proofs[4] [5]: interactive proofs limited to one message from prover to verifier. It is assumed that the theorem x to be proven is shared by both parties, and how it is chosen is external to the NIZK proof protocol. The prover and verifier do, however, share a common reference string: a fixed bit-string which is known to be randomly chosen.

Definition 8 *Let* L *be a language in* $\mathcal{NP}$ *with witness relation* W*. A function and pair of interactive machines* $(l, \mathtt{P}, \mathtt{V})$*, is called a* non-interactive proof system *for* L *if the machine* $\mathtt{V}$ *is polynomial time,* l *is a polynomial, and for all polynomials* q *and sufficiently large* η*:*

[4]The terminology is somewhat misleading. A non-interactive proof is in fact a a special type of interactive proof. It is called a *non-interactive* proof because there is only one message exchanged, and so the verifier has no chance to affect the behavior of the prover.

1. Completeness: *For all theorems $x \in L$ of length η, all witnesses w such that $(x, w) \in W$, and all reference strings σ of length $l(\eta)$*

$$\mathtt{V}\left(x, \mathtt{P}\left(x, w, \sigma\right), \sigma\right) = 1 - \frac{1}{q(\eta)}$$

2. Soundness: *For every $x \notin L$ and every unbounded adversary $\mathtt{A}$,*

$$\Pr\left[\sigma \leftarrow \{0,1\}^{l(\eta)};\ (x, p) \leftarrow \mathtt{A}(1^\eta) : Verifier(x, p, \sigma) = 1\right] \leq \frac{1}{q(\eta)}$$

That is, the prover should be able to make the verifier accept every true statement, and the adversary should have only a negligible chance of being able to make the verifier accept a false statement.

Recall that if $L \in NP$, then by definition there is a Turing machine $\mathtt{M}_L$ so that $x \in L$ if and only if there exists a w so that $\mathtt{M}_L$ accepts $\langle x, w\rangle$ in time polynomial in $|x|$. Such a w is called a *witness* to x. Clearly, if $L \in \mathcal{N}P$, then the Turing machine $\mathtt{M}_L$ can act as a verifier, and to prove that $x \in L$ it is enough for the prover to find a witness to x. There are, however, cases where one may wish to prove that $x \in L$ without disclosing a witness to x. There are several definitions formalizing this intuition; in this work we consider only the simplest of these definitions:

A non-interactive *zero-knowledge (NIZK)* proof [5] is a non-interactive proof with one interesting condition: that after the proof is completed, the verifier gains no information other than the fact that $x \in L$:

Definition 9 *Let $(l, \mathtt{P}, \mathtt{V})$ be a non-interactive proof system for some language $L \in \mathcal{N}P$. Then $(\mathtt{P}, \mathtt{V}, \mathtt{S} = (\mathtt{S}_1\mathtt{S}_2))$ is* computational non-interactive zero-knowledge *if $\mathtt{S}$ is a pair of PPT machines, and for all adversaries $\mathtt{A} = (\mathtt{A}_1\mathtt{A}_2)$:*

$$\begin{aligned}\Pr\left[\sigma \leftarrow \{0,1\}^{l(\eta)};\ (x, w, s) \leftarrow \mathtt{A}_1(\sigma);\ p \leftarrow \mathtt{P}(x, w, \sigma) : \mathtt{A}_2(x, p, s) = 1\right] \cong \\ \Pr\left[(\sigma, \tau) \leftarrow \mathtt{S}_1(1^\eta);\ (x, w, s) \leftarrow \mathtt{A}_1(\sigma);\ p \leftarrow \mathtt{S}_2(x, \sigma, \tau) : \mathtt{A}_2(x, p, s) = 1\right]\end{aligned}$$

That is, a proof system is zero-knowledge if, no matter how a malicious verifier $\mathtt{A}$ tries to glean more information from a prover $\mathtt{P}$, the prover could have been replaced by a simulator $\mathtt{S}$ without affecting the behavior of the adversary. Hence, anything that the malicious verifier $\mathtt{A}'$ could have gleaned from the prover, it could also have gleaned without access to the prover at all.

We will also require that the proof be *reference-string specific.* This is an apparently new condition which states that a proof that is valid under one reference string is most likely invalid under another reference string:

Definition 10 *Let* $(l, \mathtt{P}, \mathtt{V})$ *be a non-interactive proof system for some language* $L \in \mathcal{NP}$. *Then* $(l, \mathtt{P}, \mathtt{V})$ *is* reference string specific *if for all polynomials* q*, for all sufficiently large* η*, for all* $x \in L$*, for all* p *such that there exists a reference string* σ *making* $\mathtt{V}(x, p, \sigma)$ *accept:*

$$\Pr\left[\sigma' \leftarrow \{0,1\}^{l(\eta)} : \mathtt{V}(x, p, \sigma') = 1\right] \leq \frac{1}{q(\eta)}$$

This is not an especially strong condition. In fact, it is almost trivial to turn any interactive proof system into one that is reference string specific:

Theorem 11 *If non-interactive proof systems exist for a language* L*, then reference string-specific non-interactive proof systems exist for* L.

Proof: If b is a bit-string, let $b_{\langle m,n\rangle}$ be the contiguous substring of b from position m to position n, inclusive. Then, let $(l, \mathtt{P}, \mathtt{V})$ be a non-interactive proof system. Then let $(l', \mathtt{P}', \mathtt{V}')$ be the system:

- $l'(\eta) = l(\eta) + \eta$,
- $\mathtt{P}'(x, w, \sigma) = \left\langle \mathtt{P}\left(x, w, \sigma_{\langle 1, l(\eta)\rangle}\right), \sigma_{\langle l(\eta)+1, l(\eta)+\eta\rangle} \right\rangle$
- $\mathtt{V}'(x, p, \sigma) = \begin{cases} 1 & \text{if } \mathtt{V}'(x, p_{\langle 1,l(\eta)\rangle}, \sigma_{\langle 1,l(\eta)\rangle}) = 1 \text{ and } p_{\langle l(\eta)+1, l(\eta)+\eta\rangle} = \sigma_{\langle l(\eta)+1, l(\eta)+\eta\rangle} \\ 0 & \text{otherwise} \end{cases}$

That is, the new reference string length is the old reference string length plus an additional η bits. The new prover uses the first $l(\eta)$ bits of the reference string to simulate the old prover, then outputs the proof plus the last η bits of the reference string. The verifier checks the proof, using the first $l(\eta)$ bits of the reference string, and then verifies that the last η bits of the proof are the same as the last η bits of the reference string.

Suppose, then, that there were an x and p and σ so that $\mathtt{V}'(x, p, \sigma) = 1$. Then

$$\Pr\left[\sigma' \leftarrow \{0,1\}^{l(\eta)} : \mathtt{V}(x, p, \sigma') = 1\right] \leq \Pr\left[\sigma|_{(l(\eta)+1, l(\eta)+\eta)} = \sigma'|_{(l(\eta)+1, l(\eta)+\eta)}\right] = \frac{1}{2^\eta}.$$

Since $\frac{1}{2^\eta}$ vanishes faster than any polynomial in η, this system is reference-string specific. Furthermore, this system is still a non-interactive proof system, since it inherits soundness and completeness from the original proof system. Hence, this new system is a reference-string specific non-interactive proof system ■

It is also easy to see that this transformation preserves the zero-knowledge aspect of a non-interactive proof system: since the additional η bits are random and untouched by the prover, adding them to the proof reveals nothing about the underlying witness.

The last condition we need is that the proof be NIZK *proofs of knowledge* [16]. That is, a proof system where if an adversary can in fact create a proof for a new theorem $x \in L$ (for $L \in \mathcal{NP}$) then it is possible to derive from that adversary a witness for x. The derivation is done by a pair of algorithms, collectively known as the *extractor*, which use the adversary as a subroutine:

Definition 12 *Let* $(l, \mathtt{P}, \mathtt{V}, \mathtt{S})$ *be a NIZK proof system for* $L \in \mathcal{NP}$ *(with witness relation* W*). Then* $(l, \mathtt{P}, \mathtt{V}, \mathtt{S}, \mathtt{E} = (\mathtt{E}_1, \mathtt{E}_2))$ *is a* NIZK proof of knowledge *for* L *if* $\mathtt{E}_1$ *and* $\mathtt{E}_2$ *are PPT algorithms and for all polynomials* q *and for all sufficiently large* η*:*

1. *(Reference string uniformity):*

$$\Pr\left[\sigma \leftarrow \mathtt{E}_1(1^\eta) : \sigma\right] \cong \Pr\left[\sigma \leftarrow \{0,1\}^{l(\eta)} : \sigma\right]$$

2. *(Witness extractability): For all adversaries* $\mathtt{A}$*,*

$$\Pr\left[\sigma \leftarrow \{0,1\}^{l(\eta)};\ (x,p) \leftarrow \mathtt{A}(\sigma) : \mathtt{V}(x,p,\sigma) = 1\right]$$
$$-\Pr\left[(\sigma,\tau) \leftarrow \mathtt{E}_1(1^\eta);\ (x,p) \leftarrow \mathtt{A}(\sigma);\ w \leftarrow \mathtt{E}_2(\sigma,\tau,x,p) : (x,w) \in W\right] \leq \frac{1}{q(\eta)}.$$

The first condition is a technical consideration, specifying that the extractor does not significantly change the behavior of the adversary when trying to extract a witness. The second consideration is the important one, specifying that if the adversary can create a new statement x together with the "proof" that makes the verifier accept, then the extractor can use the adversary to produce the witness showing that $x \in L$ (with almost as great a chance of success).

Chapter 4

Relating the Formal and Computational Messages

A major result of this work will be the demonstration that certain kinds of computational cryptography can result in an equivalence of sorts between the formal and computational adversary. Before we can consider the adversary, however, we must first consider how formal messages correspond to the computational analogue: probability distributions on bit strings. In this chapter, we will define a function from formal messages to computational 'messages' using some of the computational primitives defined in Chapter 3. While the material of this chapter is not exactly that of Abadi and Rogaway [1] it is an exact analogue of their work, adapted from the case of symmetric encryption to that of public-key encryption.

The conversion function will use several helper functions, mostly to handle the translation of the atomic elements:

- Except for the keys in $\mathcal{K}_{Adv}$, all keys are assumed to be randomly and independently generated. Let $\mathsf{Keymaps}_\eta$ be a family of distributions indexed by $\eta \in \mathsf{Parameter}$, such that each distribution $\mathsf{Keymaps}_\eta$ is over a set of functions $\mathsf{f} : \mathcal{K} \to \mathsf{PublicKey} \cup \mathsf{PrivateKey}$. We further assume that for all $\eta \in \mathsf{Parameter}$,

$$\Pr\left[\mathsf{f} \leftarrow \mathsf{Keymaps}_\eta : \mathsf{f}(K) = x \wedge \mathsf{f}(K^{-1}) = y\right] =$$
$$\Pr\left[r \leftarrow \mathsf{Coins} : G(1^\eta, r) = (x, y)\right]$$

That is, the probability that a random element drawn from $\mathsf{Keymaps}_\eta$ assigns a given pair of bit-strings to a given pair of keys is exactly the same as the probability of getting that pair of bit-strings from the key generation algorithm. In other words, we assume that each formal key pair is assigned computational keys by the key generation algorithm, independently of every other key pair.

- The keys generated by the adversary, on the other hand, are not necessarily chosen randomly. They can be chosen in any way that the adversary likes, so long as they are actually keys. (That is, so that they are elements of Key.) We let Ψ_η be the set of all functions from keys in $\mathcal{K}_{Adv}$ to bit-strings, such that for all $\psi \in \Psi_\eta$,

  $$\forall K \in \mathcal{K}_{Pub}, \psi(K) \in \mathsf{PublicKey}$$

  and

  $$\forall K \in \mathcal{K}_{Priv}, \psi(K) \in \mathsf{PrivateKey}$$

- We will assume that there exists an injective function from names to bit-strings

  $$\mu : \mathcal{M} \to \{0,1\}^* .$$

  That is, no two names have the same representation.

- Lastly, we need a helper function to map nonces into bit-strings. Intuitively, nonces are random values created freshly as needed. Hence, the most natural translation is to map nonces into bit-strings with a length that grows with the security parameter. However, we must carefully consider nonces that can be chosen by the adversary. In those cases, the adversary may choose the nonce to have some particular property which the adversary can then exploit later. So, as we did with keys, we must consider two disjoint sets of nonces:

  - The first set of nonces, $\mathcal{R}_{Adv}$ are nonces whose values can be chosen by the adversary.
  - The second set, $\mathcal{R} \setminus \mathcal{R}_{Adv}$, are nonces chosen by regular, honest principals. We assume that the values for these nonces are chosen independently and at random.

  To handle these two cases, we define two sets:

  - We let Ω_η be the set of functions from $\mathcal{R} \setminus \mathcal{R}_{Adv}$ to bit-strings of length η, and
  - We let Π_η be the set of functions from $\mathcal{R}_{Adv}$ to bit-strings of length η.

  In "practice," regular participants choose their nonces randomly and independently, and it is assumed that the adversary picks their nonces in any way it chooses. We will represent this by using a random element of Ω_η, while considering all possible elements of Π_η.

With these helper functions, we can define a function that maps formal messages onto probability distributions on bit-strings:

Definition 13 *Fix an η. If $\mathsf{f} \in \mathsf{Keymaps}_\eta$, $\omega \in \Omega_\eta$, $\pi \in \Pi_\eta$ and $\psi \in \Psi_\eta$, then we define* `Convert` : $\mathcal{A} \to \mathcal{D}(\{0,1\}^*)$*, a function from messages to probability distributions on bit-strings, in the following way:*[1]

$$\texttt{Convert}_{\mathsf{f},\omega,\pi\mu}(M) =$$
$$\begin{cases} \langle \omega(M),\ \text{“nonce”} \rangle & \text{if } M \in \mathcal{R} \setminus \mathcal{R}_{Adv} \\ \langle \pi(M),\ \text{“nonce”} \rangle & \text{if } M \in \mathcal{R}_{Adv} \\ \langle \mu(M),\ \text{“name”} \rangle & \text{if } M \in \mathcal{M} \\ \langle \psi(M),\ \text{“key”} \rangle & \text{if } M \in \mathcal{K}_{Adv} \\ \langle \mathsf{f}(M),\ \text{“key”} \rangle & \text{if } M \in \mathcal{K} \setminus \mathcal{K}_{Adv} \\ \langle \mathcal{E}\left(\texttt{Convert}_{\mathsf{f},\omega,\pi,\mu}(M'), U\left[\mathsf{Coins}\right], f(K')\right),\ \text{“enc”} \rangle & \text{if } M = \{\!|M'|\!\}_{K'} \\ \langle \texttt{Convert}_{\mathsf{f},\omega,\pi\mu}(M'), \texttt{Convert}_{\mathsf{f},\omega,\pi\mu}(M''),\ \text{“pair”} \rangle & \text{if } M = M'\,M'' \end{cases}$$

(Here, we interpret the pair (encryption) operation as a function from pairs (triples) of distributions to distributions, constructed in the intuitive way.) We will denote

$$\texttt{Convert}_{\mathsf{f},\omega,\pi,\mu}(M) = [\![M]\!]$$

(The f, ω, π, ψ and μ will be implicit.)

Given this function, we can reconsider the main focus of Abadi and Rogaway: the conditions under which two formal messages should “look” the same to the formal adversary. A formal adversary has the power to make certain, limited deductions from formal messages; two given formal messages should “look” the same when all possible deductions that can be made about them yield the same results. For example, if the adversary has no other information, the two messages

$$\left\{\!\left|\{\!|A|\!\}_{K_2}\ B\right|\!\right\}_{K_1}\ K_1^{-1} \quad \text{and} \quad \left\{\!\left|\{\!|C|\!\}_{K_2}\ B\right|\!\right\}_{K_1}\ K_1^{-1}$$

should be indistinguishable to the formal adversary.

We represent the information that can be deduced from a formal message by its *pattern*:

Definition 14 *Let $T \subseteq \mathcal{K}$. We recursively define the function $p(M,T)$ to be:*

[1] *The notation $U\left[S\right]$ means the uniform distribution on the set S.*

- $p(K, T) = K$ *if* $K \in \mathcal{K}$
- $p(A, T) = A$ *if* $A \in \mathcal{M}$
- $p(N, T) = N$ *if* $N \in \mathcal{R}$
- $p(N_1\, N_2, T) = p(N_1, T)\, p(N_2, T)$
- $p(\{\!|M|\!\}_K, T) = \begin{cases} \{\!|p(M,T)|\!\}_K & \textit{if } K^{-1} \in T \\ \Box_K & \textit{otherwise} \end{cases}$

Then $\textit{pattern}_{pk}(M, T)$*, the* public-key pattern *of an expression* M *given the set* T *is*

$$p(M, C\,[\{M\} \cup \mathcal{K}_{Pub} \cup T]).$$

The grammar/algebra for patterns in exactly that of messages, with the addition of $\Box_K$ (a "blob" of K) to represent undecipherable encryptions. The blob is labeled because although the formal adversary cannot deduce anything about the plaintext, it is still able to deduce the key used to encrypt.

The main result of Abadi and Rogaway is that under their `Convert` algorithm, if two formal messages have the same pattern, then the distributions they induce are computationally indistinguishable to all passive adversaries. This is, however, true only under a technical condition[2]:

Definition 15 *For an expression* M*, construct a graph* G_M *where the nodes are the public/private key pairs used in the expression. We draw an edge from* $p_1 \to p_2$ *if in* M *the private key* K_2^{-1} *associated with pair* p_2 *is encrypted with* K_1*, the public key associated with* p_1*. The expression* M *is* acyclic *if the graph* G_M *is acyclic.*

That is, an expression M is acyclic if, when K_1 encrypts K_2^{-1}, and K_2 encrypts K_3^{-1}, and so on, this sequence of keys encrypting keys never loops back on itself. For example, the message

$$\{\!|K_1^{-1}|\!\}_{K_2}\ \{\!|K_2^{-1}|\!\}_{K_3}$$

is acyclic, but the message

$$\{\!|K_1^{-1}|\!\}_{K_2}\ \{\!|K_2^{-1}|\!\}_{K_1}$$

is not.

The result of Abadi and Rogaway remains true under the new `Convert` function:

[2]The Abadi and Rogaway version of this condition is with regards to symmetric keys.

Theorem 16 *Suppose M and N are acyclic expressions. If $pattern_{pk}(M,T) = pattern_{pk}(N,T)$ for some $T \subseteq \mathcal{K}$, then $[\![M]\!] \cong [\![N]\!]$.*

That is, the new `Convert` algorithm provides the same security against a passive distinguisher as Abadi and Rogaway's `Convert` algorithm for symmetric-key cryptography. The proof is exactly that of Abadi and Rogaway's, and is in Appendix B for brevity. The proof also yields another interesting fact:

Corollary 17 *For any $T \subseteq \mathcal{K}$ and any acyclic message M, $[\![M]\!] \cong [\![pattern_{pk}(M,T)]\!]$.*

In other words, if we assign a distribution to the *pattern* of a message (by extending Definition 14 to assign a distribution to $\Box_K$), then every message is indistinguishable from its pattern. The proof for this, also, is given in Appendix B.

The above definition concerns acyclic messages. We will later need to apply the above results to *sets* of messages. To do that, we need one intermediate operator that transforms a set of messages into a single message:

Definition 18 *Let S be a set of messages. Then, let $\overline{S}$, the* flattening *of S, be the* message *created by applying the* pair *operation to every element in S in some canonical order.*[3]

With this, we can define an acyclic set of messages:

Definition 19 *A set S of messages is acyclic if $\overline{S}$ is an acyclic message.*

This definition will become useful in the proof of our main result (Theorem 26). Also, it is entirely reasonable that the messages available to the adversary as the result of any realistic protocol will be acyclic. Usually, actual cryptographic protocols operate in one of three ways:

- Long-term keys are used to encrypt session keys, which themselves never encrypt other keys,
- The present session key is used to encrypt the next session key, but never the previous, or
- Keys are never encrypted at all.

Cycles of keys in encryptions will never result in any of these cases. Hence, in "real-world" protocols, no set of messages sent by regular participants will result in a key-cycle.

[3] *For example, if $S = \{A, B, C\}$, then $\overline{S} = A\,B\,C$.*

Chapter 5

The Active Synthesizer

As opposed to the adversary considered in [1], the adversary of interest to us is an active synthesizer eavesdropping on a fixed protocol. In this chapter, we will discuss the additional powers of this adversary.

Both Theorem 16 and Corollary 17 address the issue of how much of a message's structure an adversary can determine given *only* the message itself and a limited set of keys. Such an adversary is even more limited than an evesdropper: while an eavesdropper will know the source of a message, the adversary of [1] knows nothing but the messages and keys.

The adversary that we will consider, however, is an *active* one, which implies two things:

1. It assumed to be listening to the traffic of a fixed and known protocol, and
2. It is able to send messages of its own devising as part of that traffic.

Because it views traffic and not simply messages, it is able to learn a great deal about a message through external channels such as traffic analysis. For example, our adversary is able to "know" that a given overheard message is the first message of a run of the Needham–Schroeder protocol (Chapter 1.1), and hence that the message will have the structure "a pair made up of A's identifier and a random nonce created by A freshly for this run, which is encrypted in B's public key". The adversary [1], on the other hand, is defined in the absence of a protocol. Hence, it has no access to the "context" of a message, and so cannot drawn such a conclusion.

How do we model the information an active adversary may glean from traffic analysis? It may be that traffic analysis may yield only partial information about the structure of messages. To fully capture this ability of the adversary, however, we will give to it *perfect* knowledge about the structure of messages. On the other hand, we will want give it this additional knowledge without giving it any knowledge about the *values* those messages contain.

To reveal the structure of a message without revealing any of its values, we will assume that every message also comes with a *tag*: a representation of the structure of the message. We define[1] a function which maps messages to machine-readable tags in the following way:

Definition 20 *Let $t(\cdot) : \mathcal{A} \to \{0,1\}^*$ be the function where:*

- *For $N \in \mathcal{M}$, $t(N) = \langle \text{“name”}, \mu(N) \rangle$ where μ is the same function from names to bits used in the `Convert` algorithm.*
- *For $N \in \mathcal{R}$, $t(N) = \langle \text{“nonce”}, l_n(N) \rangle$ where $l_n(N)$ produces a machine-readable label uniquely identifying the nonce N, where $l(N)$ is completely independent of the distribution $[\![N]\!]$.*
- *For $K \in \mathcal{K}$, $t(N) = \langle \text{“key”}, l_k(N) \rangle$ where $l_k(N)$ produces a machine-readable label uniquely identifying the key K, where $l_k(K)$ is completely independent of the distribution $[\![K]\!]$.*
- *For the pair term $M\,N$, $t(M\,N) = \langle \text{“pair”}, t(M), t(N) \rangle$*
- *For the encryption term $\{\!|M|\!\}_K$, $t(\{\!|M|\!\}_K) = \langle \text{“enc”}, t(M), t(K) \rangle$*

For example, the tag of the message $\{\!|A\,N_a|\!\}_{K_B}$ (the first message in the Needham-Schroeder protocol) would be:

$$t\left(\{\!|A\,N_a|\!\}_{K_B}\right) = \Big\langle \text{“enc”}, \big\langle \text{“pair”} \left\langle \text{“name”}, \mu(A) \right\rangle, \left\langle \text{“nonce”}, l_n(N_a) \right\rangle \big\rangle, \left\langle \text{“key”}, l_k(K_B) \right\rangle \Big\rangle$$

The primary difference between a tag and a pattern is twofold: first, the tag reveals the entire structure of a message while a pattern may reveal only partial information, and a tag does not contain any of the values of a message, while a pattern reveals values to extent that it reveals structure.

We will actually use tags in two different contexts, but in the same essential way in both. The tag represents the internal structure of a message. In the first context, discussed above, we will use it to provide to the adversary knowledge about internal structure of messages that it could possibly gain from traffic analysis. In particular, we will simply assume that every message overheard by the adversary is also tagged in the above way, and will include the tags as part of the adversary's input. This is in direct contradiction to the spirit of the work of Abadi and Rogaway [1], which explores the cases in which the structure of an encryption is hidden from the adversary. We do not contradict their results, but we regard the structure of overheard encryptions as predictable in many cases and so make that information explicit.

[1]Because Abadi and Rogaway were interested only in passive adversaries, they has no need to define tags in their work. This definition, as with the rest of this chapter, are original to this work.

In the second context, we will use tags to discuss (in the abstract) how honest participants decide if a given message has a certain structure. The `Convert` algorithm provides a way to turn a formal term into a distribution of bit-strings. But suppose a protocol participant receives a bit-string. To what formal term does the message correspond? How can a protocol participant decide this? We resolve this difficulty with the idea of a *conversion verifier*:

Definition 21 *Let* $l_1 = (\omega \cup \pi) \circ l_n^{-1}$ *be a function from nonce tags to nonce bit-strings, and* $l_2 = (\mathsf{f} \cup \psi) \circ l_k^{-1}$ *be a function from key tags to key bit-strings. Then we say that the algorithm* C *is a* conversion verifier *if the verifier* C *accepts all legitimate encodings of a message. That is, for all messages* M*, for all* $m \in [\![M]\!]$*,* $\mathsf{C}^{f_\eta(\cdot),\omega(\cdot),\pi(\cdot),\psi(\cdot),l_1(\cdot),l_2(\cdot)}(m, t(M)) = 1$.

In contexts where they are clear, we will omit the oracles $f_\eta(\cdot)$, $\omega(\cdot)$, $l_1(\cdot)$, and $l_2(\cdot)$.[2] The conversion verifier will be useful in defining the challenge faced by a synthesizing adversary. Informally, such an adversary is charged with creating a new, valid message. When we formalize the adversary, however, we will use the conversion verifier to decide when a message is "valid." It is possible that the conversion verifier will accept strings which are not valid encodings. A major result of the work, Theorem 26, is that a conversion verifier can be constructed so that it is very hard to find such a string.

Before we formalize the challenge faced by the active synthesizer, however, we formalize the active synthesizer itself. As usual, the adversary is modeled as a probabilistic polynomial time Turing machine `A`. However, as an active adversary, it can send messages to honest participants and receive their replies. We represent this — and its access to other information not in the input — by giving it access to potentially helpful oracles:

- The adversary is allowed to know the public key of any principal, which we represent by way of an oracle:
  - `oracle PubKeyOf`(x)
    - `returns` $f(K_X)$ `if` $\mathsf{C}(x, t(X))$ `and` $X \in \mathcal{M}$
    - `returns` $\perp$ `otherwise`

  Since no two names have the same representation (that is, $[\![M]\!] \neq [\![N]\!]$ for all M, $N \in \mathcal{M}$), this oracle is well-defined. (In this and the next oracle, we assume that the f picked by `Initialize` is the same f used by all oracles.)

[2] Note that the conversion verifier may have more power than we conceive of formal participants as having. We will deal with this issue in another paper; for our present purposes, we will regard the environment as knowing every secret.

- Also, we assume that the adversary is *static*, in the sense that the adversary has corrupted a fixed and unchanging collection of principals. As opposed to *dynamic* adversaries, who can defer their decision as to who to corrupt until they have gathered some information, our adversary must operate with a fixed set of corrupted participants which is chosen at the beginning.

  We formalize this by assuming that there is a set of principals $\mathit{Subv} \subseteq \mathcal{M}$ that are not trustworthy. Either they *are* the adversary, or have been subverted by the adversary. The exact circumstances do not matter; what matters is that the possessors of the keys of those in *Subv* can deviate from the Needham–Schroeder protocol. The set *Subv* will not change over the course of the protocol: although the adversary may learn new keys due to flaws in the protocol (which we handle through the closure operation) it cannot learn new keys by subverting new principals.

  We represent corruption of principals in *Subv* by allowing the adversary oracle access to their private keys:

  – `oracle PrivKeyOf`(x)

    * `returns` $f(K_X^{-1})$ `if` $\mathsf{C}\,(x, t\,(X))$ `and` $X \in \mathit{Subv}$
    * `returns` $\perp$ `otherwise`

- Lastly, the adversary should be able to know the name of every participant and which ones it has subverted. We can explicitly model this with an oracle that provides names of participants upon some sort of request, but this is not necessary. There is nothing secret or unpredictable about the naming of participants. It is easy to imagine that the naming of participants can be done by an efficient deterministic algorithm which can be "hard-wired" in to the adversary. Hence, we can assume, without loss of generality, that the adversary knows the names of all relevant participants.

In the next chapter, we discuss the challenge of creating a valid-appearing message, and the equivalence between the formal attacks and the active synthesizer.

Chapter 6

Relating Formal Attacks and the Active Synthesizer

In this chapter, we explore the relationship between formal adversaries and active synthesizers. In particular, we define the concept of an "attack" on both the computational and formal levels, and show that under certain conditions almost every computational attack will correspond to a formal attack.

Formal attacks have already been defined in Definition 4. We define a computational attack to be the activity of an active synthesizer. However, as mentioned in Chapter 2, we assume that each message sent by the adversary is "expected" by the environment. In future work we will refine this notion; for the time being avoid this particular issue but considering a more conservative goal. Rather than requiring the adversary to synthesize an element of some particular set of messages (the set of "expected" messages), we will consider the more general case where the adversary must only synthesize any message at all. Hence, the only requirement that we will consider here is that every message sent by the active synthesizer be an apparently-valid encoding of some formal message:

Definition 22 *A* quasi-formal attack against environment *Env' (with conversion verifier C) is a trace of a computational adversary in communication with the environment Env'. That is, a sequence of bit-string queries ($\mathtt{Q}_i$) and receptions of sets of (tagged) bit-string responses ($\mathtt{R}_i$):*

$$\mathtt{R}_0 \quad \mathtt{Q}_1 \quad \mathtt{R}_1 \quad \ldots \quad \mathtt{Q}_{n-1} \quad \mathtt{R}_{n-1} \quad \mathtt{Q}_n \quad \mathtt{R}_n$$

such that for all i,

- *for some $M_i \in \mathcal{A}$, $\mathsf{C}(\mathtt{Q}_i, t(M_i))$, and*

- *for each element* $\mathtt{r} \in \mathtt{R}_i$*,* $\mathtt{r} = (\mathtt{r}', \mathtt{t}')$ *where*
  - $\mathtt{r}' \in [\![N_r]\!]$ *for some* $N_r \in \mathcal{A}$*, and*
  - $\mathtt{t}' = t\,(N_r)$

(Note that the queries $\mathtt{Q}_i$ only need to fool the conversion verifier, while the responses $\mathtt{R}_i$ actually contain legitimate encodings.) We call this type of attack quasi-formal because it is neither entirely in the computational setting nor entirely in the formal setting. It is not entirely computational since the adversary's queries must be encodings of formal messages rather than arbitrary bit-strings. The attack is not entirely formal, on the other hand, since the queries of the adversary are not limited by any sort of closure operations. By definition, however, a quasi-formal attack does have a corresponding formal sequence:

Definition 23 *A* corresponding formal sequence *of a quasi-formal attack*

$$\mathtt{R}_0 \quad \mathtt{Q}_1 \quad \mathtt{R}_1 \quad \ldots \quad \mathtt{Q}_{n-1} \quad \mathtt{R}_{n-1} \quad \mathtt{Q}_n \quad \mathtt{R}_n$$

is an alternating sequence of

- *messages in* $\mathcal{A}$ *(*q_i*) and*
- *sets of messages in* $\mathcal{A}$ *(*r_i*):*

$$r_0 \quad q_1 \quad r_1 \quad q_2 \quad r_2 \quad \ldots \quad q_{n-1} \quad r_{n-1} \quad q_n \quad r_n$$

such that for all i*,*

- $\mathsf{C}(\mathtt{Q}_i, t\,(q_i)) = 1$*,*
- *there is a bijection* f_i *from* $\mathtt{R}_i$ *to* r_i *such that if* $f(\mathtt{r}', \mathtt{t}') = N_r$ *then* $\mathtt{r}' \in [\![N_r]\!]$*.*

There may be more than one corresponding formal sequence for a quasi-formal attack, if the `Convert` algorithm allows more than one formal term to have the same encoding. Note, however, that in both of the above definitions we assume that each application of the `Convert` algorithm uses the same $\mathtt{f}$, ω, π and ψ functions.

We wish to prove that a quasi-formal attack in the computational setting must imply an attack in the formal setting. That is, we wish to show that any possible quasi-formal attack has a corresponding formal attack. However, the definition of a quasi-formal attack is a great deal weaker than that of a formal attack, and so any correspondence that we may be able to find will be a result of the cryptography used in the `Convert` algorithm. To that end, we define a security condition that limits the power of the adversary in calculating a single query:

Definition 24 *An encryption scheme* $(\mathsf{G}, \mathsf{E}, \mathsf{D}, \mathsf{C})$ *is* ideal *if*

1. $(\mathsf{G}, \mathsf{E}, \mathsf{D})$ *is a key-providing semantically secure (one-pass) encryption scheme, and*
2. *The conversion verifier* C *is such that the adversary cannot satisfy the verifier with something outside the closure:*

$$\begin{array}{ll} & \forall\, \mathtt{A}_{PPT},\ \forall\, acyclic\ S \subseteq \mathcal{A},\ \forall\, M \in \left(\mathcal{A} \setminus C\left[\widehat{S}\right]\right),\ \forall\, polynomials\ q,\ \forall\ sufficiently\ large\ \eta, \\ & \quad \forall\, \pi \in \Pi_\eta,\ \forall\, \psi \in \Psi_\eta : \\ \Pr[& f_\eta \leftarrow \mathsf{Keymaps}_\eta; \\ & \omega \leftarrow \Omega_\eta; \\ & s \leftarrow [\![S]\!]; \\ & m \leftarrow \mathtt{A}^{\mathit{PubKeyOf}(\cdot),\mathit{PrivKeyOf}(\cdot)}(1^\eta, s, t(S)) : \\ & \mathsf{C}(m, t(M)) = 1 \qquad\qquad\qquad\qquad] \leq \frac{1}{q(\eta)} \end{array}$$

That is, the encryption is ideal if it is semantically secure and the adversary cannot make anything that seems to be the encoding of something outside the closure. Note that this definition applies only to an *acyclic* set A of messages; although it would be natural to extend it to all sets, we need the acyclicity property for our proofs.

Despite the fact that this condition describes only one query, it is sufficient to enforce the correspondence between quasi-formal and formal cryptography:

Theorem 25 *Let* $\left\langle \mathtt{A}_i^{\mathit{PubKeyOf}(\cdot),\mathit{PrivKeyOf}(\cdot)}, Env' \right\rangle$ *be the distribution on communication transcripts between an interactive Turing machine* $\mathtt{A}_i$ *(with oracle access to* $\mathit{PubKeyOf}(\cdot)$ *and* $\mathit{PrivKeyOf}(\cdot)$*) and the environment* Env'*. If the* **Convert** *algorithm uses ideal encryption, and the number of messages in the exchange generated by the adversary is some fixed number* n*, then*

$$\Pr\left[c \leftarrow \left\langle \mathtt{A}_i^{\mathit{PubKeyOf}(\cdot),\mathit{PrivKeyOf}(\cdot)}(1^\eta), Env' \right\rangle : \begin{array}{l} c\ is\ quasi\text{-}formal\ and\ has \\ no\ corresponding\ formal \\ attack \end{array} \right] \leq \frac{1}{q(\eta)}$$

for all polynomials q *and sufficiently large* η*.*

Proof: Consider the quasi-formal attack:

$$\mathtt{R}_0 \quad \mathtt{Q}_1 \quad \mathtt{R}_1 \quad \ldots \quad \mathtt{Q}_{n-1} \quad \mathtt{R}_{n-1} \quad \mathtt{Q}_n \quad \mathtt{R}_n$$

Since it is quasi-formal, it has a corresponding formal sequence:

$$r_0 \quad q_1 \quad r_1 \quad q_2 \quad r_2 \quad \ldots \quad q_{n-1} \quad r_{n-1} \quad q_n \quad r_n$$

One way to consider this is as n attempts on the part of the adversary to break the ideal encryption. In particular, let $u_i = \bigcup_{j=0}^{i} r_i$. Then query q_i can be considered to be the i attempt to produce a

$$q_i \notin C\left[\widehat{u_{i-1}}\right]$$

At each stage, the interactive adversary $\mathtt{A}_i$ has received (as input) some $r' \leftarrow [\![u_{i-1}]\!]$ and the corresponding tag $t' = t\left(u_{i-1}\right)$. It has oracle access to $\mathtt{PubKeyOf}(\cdot)$ and $\mathtt{PrivKeyOf}(\cdot)$, and runs in time polynomial in η. Hence, we know by the definition of ideal encryption that

$$\Pr\left[q_i \notin C\left[\widehat{u_{i-1}}\right]\right] = f_i(\eta) \leq \frac{1}{q(\eta)}$$

for all polynomials q and sufficiently large η. Hence we can use a union bound to upper bound

$$\Pr\left[\exists i\ .q_i \notin C\left[\widehat{u_{i-1}}\right]\right] \leq \sum_{i=1}^{n} f_i(\eta) \leq \frac{n}{q(\eta)}$$

for all c and all sufficiently large η. Since n is constant with respect to η, the probability that the adversary can produce a quasi-formal attack that is not in fact formal remains negligible. ■

Hence, in order to show that the computational adversary is limited to the operations available to the formal adversary, it is sufficient to use ideal cryptography. Can ideal cryptography be achieved? Yes, quite easily, a fact that we demonstrate in the next chapter.

Chapter 7

Ideal Encryption

In this chapter, we show that a specific encryption scheme is ideal:

Theorem 26 *Suppose that* $(\mathsf{G}', \mathsf{E}', \mathsf{D}')$ *is a key-providing, semantically secure (one-pass) encryption scheme. Let* $O(\cdot)$ *be the random oracle, and suppose that* $(l, \mathtt{P}, \mathtt{V}, \mathtt{S}, \mathtt{E})$ *is a reference-string specific, non-interactive zero-knowledge proof of knowledge system for* Ciphertext. *Let:*

- $\mathsf{G} = \mathsf{G}'$
- $\mathsf{E}_k(m)$ *be the distribution:*

$$c \leftarrow \mathsf{E}'_e(m);\ p \leftarrow \mathtt{P}(c, \langle m, r, e \rangle, O(c)_{\langle 1, l(\eta) \rangle}) : \langle c, p \rangle$$

  (where r *is the randomness used by* $\mathsf{E}'_e(m)$ *to produce* c*),*

- $\mathsf{D}(\langle c, p \rangle, d) = \begin{cases} m & \text{if } \mathsf{D}(c,d) = m \text{ and } \mathtt{V}(c, p, O(c)_{\langle 1, l(\eta) \rangle}) = 1 \\ \perp & \text{otherwise} \end{cases}$ *, and*
- $\mathsf{C}(x, t)$ *be defined recursively:*
  - *If* $t = \langle \text{“nonce”}, n \rangle$ *and* $x = \langle l_1(n), \text{“}\mathtt{nonce}\text{”} \rangle$*, then accept.*
  - *If* $t = \langle \text{“key”}, k \rangle$ *and* $x = \langle l_2(k), \text{“}\mathtt{key}\text{”} \rangle$*, then accept.*
  - *if* $t = \langle \text{“pair”}, m, n \rangle$ *and* $x = \langle a, b, \text{“}\mathtt{pair}\text{”} \rangle$*, then accept if* $\mathsf{C}(m, a)$ *and* $\mathsf{C}(n, b)$ *accepts.*
  - *If* $t = \langle \text{“enc”}, t, k \rangle$*, and* $x = \langle \langle c, p \rangle, \text{“}\mathtt{enc}\text{”} \rangle$ *then accept if and only if:*
    1. *The verifier* $\mathtt{V}(c, p, O(c)_{\langle 1, l(\eta) \rangle}) = 1$*,*
    2. $\mathsf{D}\left(\langle c, p \rangle, (l_2(k))^{-1}\right) = m$*, and*
    3. $\mathsf{C}(m, t)$ *accepts.*

(Recall that l_1 and l_2 are functions from nonce tags and key tags to nonce encodings and key encodings, respectively.)

Then $(\mathsf{G}, \mathsf{E}, \mathsf{D}, \mathsf{C})$ *is ideal.*

Proof:

To show that this collection of algorithms satisfies the definition of ideal cryptography, we need to show that they provide semantic security, and that the conversion verifier cannot be fooled. We show that the algorithms above provide semantic security in Appendix C focus our attention here on the second condition of the definition.

Suppose there is an adversary that violates the condition. That is,

$$\exists\, \mathtt{A},\ \exists \text{acyclic } S \subset \mathcal{A},\ \exists\, M \notin \left(\mathcal{A} \setminus C\left[\widehat{S}\right]\right),\ \exists \text{ a polynomial } q,\quad \text{for infinitely many } \eta,\ \exists\, \pi \in \Pi_\eta,\ \exists\, \psi \in \Psi_\eta$$

$$\begin{aligned}\Pr[\quad & f_\eta \leftarrow \mathsf{Keymaps}_\eta; \\ & \omega \leftarrow \Omega_\eta; \\ & s \leftarrow [\![S]\!]; \\ & m \leftarrow \mathtt{A}^{\mathtt{PubKeyOf}(\cdot),\mathtt{PrivKeyOf}(\cdot)}(1^\eta, s, t(S)) : \\ & \mathsf{C}(m, t(M)) = 1 \qquad\qquad] \geq \tfrac{1}{q(\eta)}\end{aligned}$$

Note that we can assume, without loss of generality, that S contains only simple terms: only atomic terms or encryptions. If S did contain a pair term $(X\,Y)$, then one could simply consider the "equivalent" set $(S \setminus \{(X\,Y)\}) \cup \{X, Y\}$. Likewise, we can again assume without loss of generality that M is a simple term: If the adversary can produce something in $[\![X\,Y]\!]$, then it is a simple matter to modify the adversary to produce something in $[\![X]\!]$ and something in $[\![Y]\!]$. And since $X\,Y \notin C\left[\widehat{S}\right]$ implies that either $X \notin C\left[\widehat{S}\right]$ or $Y \notin C\left[\widehat{S}\right]$, it is sufficient to consider only an adversary that produces simple terms.

Then we proceed by case analysis:

- Suppose that $M = \{\!|N|\!\}_K$. In that case, m is equal to $\langle\langle c, p\rangle, \text{“}\mathtt{enc}\text{”}\rangle$ where
  - c is the output of some GM-secure encryption scheme E', and
  - p is a non-interactive, zero-knowledge reference string-specific proof of knowledge, proving knowledge of the plaintext associated with c above (and the randomness used in the encryption). The reference string for this proof is assumed to be the first $l(\eta)$ bits of $O(c)$, where O is a random oracle.

  So, suppose that the adversary is able to produce, with polynomial probability, a pair $\langle c, p\rangle$, and $\langle\langle c, p\rangle, \text{“}\mathtt{enc}\text{”}\rangle \in [\![\{\!|N|\!\}_K]\!]$. Since $\mathsf{C}(\langle\langle c, p\rangle, \text{“}\mathtt{enc}\text{”}\rangle, t(M))$ accepts, it must be that $\mathtt{V}(c, p, O(c)_{\langle 1, l(\eta)\rangle})$

accepted also.

Since p is a proof of knowledge, we know from Definition 12 that there is an extractor $\mathtt{E} = (\mathtt{E}_1, \mathtt{E}_2)$ such that

$$\Pr\left[\sigma \leftarrow \mathtt{E}_1(1^\eta) : \sigma\right] \cong \Pr\left[\sigma \leftarrow \{0,1\}^{l(\eta)} : \sigma\right]$$

and if

$$\begin{aligned}&\Pr\left[(\sigma,\tau) \leftarrow \mathtt{E}_1(1^\eta);\ (x,p) \leftarrow \mathtt{A}(\sigma);\ w \leftarrow \mathtt{E}_2(\sigma,\tau,x,p) : (x,w) \in W\right] \\ &\quad = \Pr\left[\sigma \leftarrow \{0,1\}^{l(\eta)};\ (x,p) \leftarrow \mathtt{A}(\sigma) : \mathtt{V}(x,p,\sigma) = 1\right] - \alpha(\eta).\end{aligned}$$

then $\alpha(\eta)$ is a negligible function (i.e., one that disappears faster than any polynomial.) Hence, if $\mathtt{A}$ produces an m that satisfies $\mathsf{C}(m, t\left(\{\!|N|\!\}_K\right))$, then there exists an $\mathtt{A}'$ that can produce an n that satisfies $\mathsf{C}(n, t\left(N\right))$:

- $\mathtt{A}'$ runs the first part of the extractor $\mathtt{E}_1$ to produce a reference string σ.
- It then runs $\mathtt{A}$, passing on any oracle queries made by $\mathtt{A}$ to its own oracles. The only exception is that when $\mathtt{A}$ tries to query the random oracle $O(\cdot)$ on c, $\mathtt{A}'$ returns σ instead of $O(c)$. $\mathtt{A}$ produces a "proof" p'.
- $\mathtt{A}'$ then uses the second part of the extractor $\mathtt{E}_2$ on $\langle c, p', \sigma\rangle$ to extract a witness w. The witness will be, of course, $\langle n, p, e\rangle$ where, n is the plaintext, r is the randomness used in the encryption algorithm, and e is the public key used to encrypt.

What are the odds that $\mathtt{A}'$ will succeed at this task? Since we know that the strings produced by $\mathtt{E}_1$ are computationally indistinguishable from random reference strings (by the first part of Definition 12) the probability that $\mathtt{A}$ will produce a seemingly-valid encoding of $\{\!|N|\!\}_K$ will not change by a non-negligible amount. Hence, we can rest assured that $\mathtt{A}'$ will still produce a theorem and proof with non-negligible advantage. In fact, since $\mathtt{A}$ had a non-negligible chance of producing a *particular* theorem before (namely c, a valid-seeming encoding of $\{\!|M|\!\}_K$), it has almost the same chances of doing so with the new reference string.

Hence, by the second part of Definition 12, we know that the probability of that he second part of the extractor $\mathtt{E}_2$ producing a witness cannot be non-negligibly less than the chances of $\mathtt{A}$ producing a proof. Hence, if the advantage of $\mathtt{A}$ in creating a proof can be bounded below by $\frac{1}{q(\eta)}$ for some polynomial q, then the advantage of $\mathtt{A}'$ in producing a witness can be bounded below by

$$\frac{1}{q(\eta)} - \alpha(\eta) - \beta(\eta)$$

where α and β are negligible functions. Hence, $\mathtt{A}'$ must have a non-negligible chance of creating a witness, which will yield the plaintext of c immediately.

One last point. How do we know that $\mathtt{A}$ will ever query the random oracle for the value $O(c)$? Because the proof is reference-string specific, there are vanishingly few reference strings that would make $\mathtt{V}$ accept p as a proof for c. Hence, if the adversary created c and p without querying the random oracle on c, then the adversary was able to produce a c such that its value, under the random oracle, happened to be one of the vanishingly few reference strings for p. Since the random oracle produces random strings, the probability of the adversary being able to find such a c is:

$$\Pr\left[\sigma' \leftarrow \{0,1\}^{l(\eta)} : \mathtt{V}(c,p,\sigma') = 1\right] \leq \frac{1}{q(\eta)}$$

Hence, if the adversary can create a $\langle c, p\rangle$ that the conversion verifier accepts, it must have used the value $O(c)$ in doing so.

Hence, if an adversary can produce a valid-seeming encryption, then it can produce the plaintext for that encryption with almost the same chances. Likewise, if the adversary can create the encoding of a pair, it can create the encoding of each component.

Now, switch tacks for a moment, and consider the parse tree of M. Suppose that every path from the root of the parse tree to a leaf passes through an element of $C\left[\widehat{S}\right]$. That is, suppose that for every path in the parse tree from root to leaf must have a node whose message is in $C\left[\widehat{S}\right]$. Then it must be that the root message, M, is in $C\left[\widehat{S}\right]$. Hence, there must be some path in the parse tree of M such that no element along that path in is $C\left[\widehat{S}\right]$, including the root M_R. We have shown above that if the adversary can create a valid-seeming encoding of M, then it can create a valid seeming encoding of M_R.

Now, consider the possibilities for M_R:

- Suppose $M_R \in \mathcal{M}$. Then $M_R \in C\left[\widehat{S}\right]$, no matter what S is. Hence, a contradiction.
- Suppose $M_R \in \mathcal{K}_{Pub}$. Then, once again, $M_R \in C\left[\widehat{S}\right]$, no matter what S is.
- Suppose $M_R \in \mathcal{R} \cup \mathcal{K}_{Priv}$. If $M_R \in \mathcal{R}_{Adv} \cup \mathcal{K}_{Subv} \cup \mathcal{K}_{Adv}$, then $M_R \in C\left[\widehat{S}\right]$. So, we only need to worry about $M_R \in \mathcal{R} \setminus \mathcal{R}_{Adv}$ or $M_R \in \mathcal{K}_{Priv} \setminus (\mathcal{K}_{Subv} \cup \mathcal{K}_{Adv})$. There are two cases: either M_R is in the parse tree of something in S, or it is not.

  If M_R is not in the parse tree of any element of S, then the input to the adversary is completely independent of the required output. Because ω is a function to which the conversion verifier has access, there is only one value m which will make $\mathsf{C}(m, t(M_R))$ accept: $\omega(M_R)$, the sole element of $[M_R]$. And because ω is drawn randomly from Ω_η and f is drawn randomly from $\mathsf{Keymaps}_\eta$, the element of $[M_R]$ is independent from the value of the encoding of any other atomic value. In particular, the value in $[M_R]$ is independent of the encodings of elements of S, and all possible encodings of elements in $\mathcal{M}$, $\mathcal{K}_{Pub}$, $\mathcal{K}_{Subv}$, and $\mathcal{R}_{Adv}$. If $M_R \in \mathcal{R}$, then the adversary in question is able to guess a η-bit random value based on inputs that are independent of the target value. The probability of this must be bounded above by $2^{-\eta}$, contradicting our assumption that the probability is polynomial. If, on the other hand, $M_R \in \mathcal{K}_{Priv}$, then the adversary is able to guess a private key based on the corresponding public key and values independent of the private key. Since we are assuming that the encryption scheme is semantically secure, the probability of this cannot be bounded below by a polynomial fraction. Hence, no matter what M_R is, the probability of the adversary being able to produce it cannot be more than a polynomial fraction.

On the other hand, it may be that M_R is in the parse tree of one or more elements of S. Then if M_R is in the parse tree of some element of S, then M_R is in the parse tree of $\overline{S}$, the flattening of S.

Suppose that M_R is in the parse tree of the $pattern_{pk}\left(\overline{S}, \mathcal{K}_{Bad}\right)$. Then M_R is in $C\left[\widehat{S}\right]$ and we reach a trivial contradiction. So, the only case of interest is that in which M_R is not in the parse tree of $pattern_{pk}\left(\overline{S}, \mathcal{K}_{Bad}\right)$. Since S is acyclic, we can apply the Abadi-Rogaway-like result of Appendix B, and instead of giving the adversary $\mathtt{A}$ a sample $s \leftarrow [S]$, we can give the adversary a sample $s' \leftarrow \llbracket pattern_{pk}\left(\overline{S}, \mathcal{K}_{Bad}\right)\rrbracket$. From Theorem 16, we know that $\llbracket S \rrbracket$ and $\llbracket pattern_{pk}\left(\overline{S}, \mathcal{K}_{Bad}\right)\rrbracket$ are computationally indistinguishable. Hence, the adversary $\mathtt{A}$ has as great a chance of being able to extract $[M_R]$ from $\llbracket pattern_{pk}\left(\overline{S}, \mathcal{K}_{Bad}\right)\rrbracket$ as it does from $[S]$. But since the distribution $\llbracket pattern_{pk}\left(\overline{S}, \mathcal{K}_{Bad}\right)\rrbracket$ is independent of the value $[M_R]$, we know that the odds of any algorithm being able to do this are negligible.

However, there is one small complication. We only know that the two distributions $[S]$ and $\llbracket pattern_{pk}\left(\overline{S}, \mathcal{K}_{Bad}\right)\rrbracket$ are indistinguishable to a PPT algorithm. Our adversary $\mathtt{A}$ is a PPT algorithm with access to additional oracles. Will the two distributions still be indistinguishable to *this* adversary?

The answer is, of course, that they are. Suppose to the contrary that our adversary $\mathtt{A}$ can distinguish between $\mathtt{A}$ a sample $s \leftarrow [S]$ and $s' \leftarrow \llbracket pattern_{pk}\left(\overline{S}, \mathcal{K}_{Bad}\right)\rrbracket$. That is, if

$$P_1^m = \Pr\left[s \leftarrow \llbracket S \rrbracket : Adv^{\mathtt{PubKeyOf}(\cdot),\mathtt{PrivKeyOf}(\cdot)}(1^\eta, s, t(S)) = m\right]$$

and

$$P_2^m = \Pr\left[s' \leftarrow \llbracket pattern_{pk}\left(\overline{S}, \mathcal{K}_{Bad}\right)\rrbracket : Adv^{\texttt{PubKeyOf}(\cdot),\texttt{PrivKeyOf}(\cdot)}(1^\eta, s', t(S)) = m\right]$$

then suppose that for some m, some polynomial q, and infinitely many η,

$$|P_1^m - P_2^m| \geq \frac{1}{q(\eta)}$$

Then we will build another adversary $\mathtt{A}'$ that distinguishes between two distributions without the additional oracles. However, the two distributions distinguished by $\mathtt{A}'$ will not be $\llbracket\overline{S}\rrbracket$ and $\llbracket pattern_{pk}\left(\overline{S}, \mathcal{K}_{Bad}\right)\rrbracket$, but a slight modification of them.

The essential idea is that we will create a new message, $\mathtt{A}'$ will use $\mathtt{A}$ to distinguish between it and its pattern. However, $\mathtt{A}$ will not distinguish between them directly; it will distinguish between a *sub*-message and *its* pattern. The parts of the message not included in the sub-message will be used by $\mathtt{A}'$ to simulate the oracles should $\mathtt{A}$ make any queries.

Let $\mathcal{K}|_M$ be the set of every key, public or private, which is either in the parse tree of M or whose inverse is in the parse tree of M. That is, $\mathcal{K}|_M$ is every key that might be useful in the analysis of M. Then let

$$\mathcal{K}(M) = \mathcal{K}|_M \cap (\mathcal{K}_{Pub} \cup \mathcal{K}_{Bad}),$$

be the set of every key available to the adversary which would be potentially useful in the analysis of M. The two distributions which our new adversary $\mathtt{A}'$ will be able to distinguish will be

$$D_1 = \left\llbracket\overline{S \cup \mathcal{K}\left(\overline{S}\right)}\right\rrbracket$$

and

$$D_2 = \left\llbracket pattern_{pk}\left(\overline{S \cup \mathcal{K}\left(\overline{S}\right)}, \mathcal{K}_{Bad}\right)\right\rrbracket$$

In essence, we add to S and $pattern_{pk}\left(S, \mathcal{K}_{Bad}\right)$ all the useful keys available to the penetrator. (We cannot simply add all the keys the adversary knows, as there may be an infinite number of them. Hence, we add only the finitely many keys may be useful in the analysis of this particular message.) By Corollary 17, we know that these two distributions should be indistinguishable.

We now construct an adversary $\mathtt{A}'$ that distinguishes D_1 from D_2, given that $\llbracket S\rrbracket$ is distinguishable from $\llbracket pattern_{pk}\left(S, \mathcal{K}_{Bad}\right)\rrbracket$ by the original adversary $\mathtt{A}$. The new $\mathtt{A}'$ operates by running $\mathtt{A}$ as a sub-routine.

Suppose that s is drawn from either D_1 or D_2. On input s, the new adversary $\mathtt{A}'$ first strips off all the keys at the ends of the string. That is, it transforms s from (for example) an element of $\left[\!\left[\overline{S \cup \mathcal{K}\left(\overline{S}\right)}\right]\!\right]$ to s', an element of $[\![S]\!]$. Then it creates a tag t. The tag of S depends only on the parse tree of S and not any of the actual values of the atomic messages at the leaves (with the exception of μ, the function from names to bits, which can be assumed to be some public, easily computable function). Hence, the parse tree and μ can be hard-wired into $\mathtt{A}'$, which can accurately generate a valid tag for both S through the function described in Definition 20.

Then the adversary $\mathtt{A}'$ simulates $\mathtt{A}^{\mathtt{PubKeyOf}(\cdot),\mathtt{PrivKeyOf}(\cdot)}$ on input $\langle 1^\eta, s', t\rangle$. This is exactly the input the original adversary $\mathtt{A}$ expects. If $\mathtt{A}^{\mathtt{PubKeyOf}(\cdot),\mathtt{PrivKeyOf}(\cdot)}$ makes an oracle query, then $\mathtt{A}'$ responds in the following way:

– If the oracle being queried is $\mathtt{PubKeyOf}(\cdot)$, and the input is a valid name N, then $\mathtt{A}'$ must "return" a key. If $K_N \in \mathcal{K}\left(\overline{S}\right)$, then the key to return was part of the original input s. Else, $\mathtt{A}'$ checks an internal table to see if a value for K_N has already been created. If so, it returns that. Otherwise, it runs $\mathtt{G}$ to create values for K_N and K_N^{-1}, stores them in its internal table, and returns the new value for K_N.

– If the oracle being queried is $\mathtt{PrivKeyOf}(\cdot)$, and the input is a valid name N, then $\mathtt{A}'$ may or may not "return" a key. If N is not a compromised participant, then $K_N^{-1} \notin \mathcal{K}_{Adv}$ and no key is returned. Otherwise, $K_N^{-1} \in \mathcal{K}_{Adv}$, and a key must be returned. If $K_N^{-1} \in \mathcal{K}\left(\overline{S}\right)$, then the key to return was part of the original input s. Else, $\mathtt{A}'$ checks an internal table to see if a value for K_N^{-1} has already been created. If so, it returns that. Otherwise, it runs $\mathtt{G}$ to create values for K_N and K_N^{-1}, stores them in its internal table, and returns the new value for K_N^{-1}.

Thus, the adversary $\mathtt{A}'$ can exactly simulate the adversary $\mathtt{A}$ by using the additional information in its input to simulate the oracles that $\mathtt{A}$ might call. Hence, the $\mathtt{A}$ subroutine will act in exactly the same fashion as it would before. If $\mathtt{A}$ produces m, then $\mathtt{A}$ returns 1. Otherwise, $\mathtt{A}$ returns 0. Since the probability that $\mathtt{A}$ produces m changes by a polynomial fraction, depending on the distribution of s', the new adversary $\mathtt{A}'$ has a polynomial advantage in predicting the distribution of its input s'. Hence, $\mathtt{A}'$ can distinguish between D_1 and D_2. Since we know that these two distributions are indistinguishable, we know that our original assumption about the original adversary $\mathtt{A}$ is in correct, and the oracles and tag give $\mathtt{A}$ no power in distinguishing between $[\![S]\!]$ and $[\![pattern_{pk}(S, \mathcal{K}_{Adv})]\!]$.

Hence, we can run $\mathtt{A}$ on a sample from $[\![pattern_{pk}(S, \mathcal{K}_{Adv})]\!]$ and achieve the same result as running $\mathtt{A}$ on a sample from $[\![S]\!]$. In both cases, the adversary has a polynomial chance of producing $[\![M_R]\!]$, even though M_R is not in that parse tree of $[\![pattern_{pk}(S, \mathcal{K}_{Adv})]\!]$. But then the distribution of $[\![pattern_{pk}\,(S, \mathcal{K}_{Adv})]\!]$ is independent of the value of $[\![M]\!]$, and the adversary $\mathtt{A}$ is able to produce a random value based only on input completely independent of that random value. Since the odds of this happening are negligible in both the case of nonces and of private keys, we know that we have reached a contradiction.

So, we have shown two things:

1. If the adversary can produce an encryption outside the closure with non-negligible probability, then there also exists an adversary that can produce the plaintext of that encryption. Hence, if an adversary can produce a compound term, then some atomic element in the parse tree of that compound term but outside $C\left[\widehat{S}\right]$ can be produced by some (potentially other) adversary.

2. The probability that an adversary can produce an atomic element outside the closure is negligible.

Hence, no adversary can produce any term, atomic or compound, outside the closure of its input with non-negligible probability. ■

The result of this theorem is that the stated encryption scheme is ideal. Thus, if the encryption scheme in the assumption is used then any quasi-formal attack (sequence which has a corresponding sequence in the formal algebra) is a formal attack (an attack available to the Dolev–Yao adversary) with a probability negligibly close to 1. That is:

Theorem 27 *Suppose that* $(\mathsf{G}, \mathsf{E}, \mathsf{D}, \mathsf{C})$ *is the encryption scheme described in Theorem 26. Then*

$$\Pr\left[c \leftarrow \left\langle \mathtt{A}_i^{\mathit{PubKeyOf}(\cdot), \mathit{PrivKeyOf}(\cdot)}(1^\eta), Env' \right\rangle : \begin{array}{c} c \textit{ is quasi-formal and has no} \\ \textit{corresponding formal attack} \end{array}\right] \leq \frac{1}{q(\eta)}$$

for all polynomials q and sufficiently large η.

Hence, if the cryptography used is that of Theorem 26, then the computational adversary cannot produce a formal message which the formal adversary could not have produced.

Chapter 8

Conclusion and Open Problems

The primary contribution paper is a rigorous computational definition of ideal cryptography (Definition 24), a concept poorly understood but widely (albeit implicitly) used in the area of formal cryptography. Simply put, our definition of ideal cryptography says that even after the computational adversary receives some collection of terms, it cannot synthesize a term that could not have been produced by the Dolev-Yao adversary. In particular, the adversary cannot create a ciphertext without having access to, or being able to synthesize with a non-negligible probability, the corresponding plaintext.

This idea may be related to the idea of "plaintext aware" encryption, due to Bellare and Rogaway ([3]). In their work, they define "plaintext aware" encryption to be (informally) encryption with the property that the adversary cannot create a ciphertext without also knowing the corresponding plaintext. Certainly, this same basic desire motivated a great deal of the construction of Theorem 26; the exact relation between plaintext-aware encryption and ideal encryption should be investigated.

Also, there are stronger possible definitions of ideal cryptography possible. For example, under our definition it may be possible that the adversary has probability 1 of making *some* new term, but not a negligible chance of making any *particular* term. For this reason, one might consider the stronger possible definition:

Definition 28 *The encryption is* superideal *if it satisfies:*

$$\begin{aligned}
&\forall\, \mathtt{A},\ \forall, S \subset \mathcal{A},\ \forall\, \textit{polynomials } q,\ \forall\, s.\,l.\,\eta,\ \forall \pi \in \Pi_\eta \\
&\Pr[\quad f_\eta \leftarrow \mathsf{Keymaps}_\eta; \\
&\qquad \omega \leftarrow \Omega_\eta; \\
&\qquad s \leftarrow [\![S]\!]; \\
&\qquad m \leftarrow \mathtt{A}^{\mathit{PubKeyOf}(\cdot),\mathit{PrivKeyOf}(\cdot)}(1^\eta, s, t\,(S)) : \\
&\qquad \exists M \notin \mathcal{A} \setminus C\left[\widehat{S}\right]\ \textit{s.t.}\ \mathsf{C}(m, t\,(M)) = 1 \qquad\qquad] \leq \tfrac{1}{q(\eta)}
\end{aligned}$$

Though it would be desirable to achieve this extremely strong definition, it is not clear if it is achievable or necessary to justify the formal model.

Another possibility is to allow the abilities of the adversary to grow with the security parameter in additional ways. In the definition of ideal cryptography (Definition 24), the running time of the adversary grows with the security parameter. However, other aspects of the problem remain constant. In particular, the set S of terms remains fixed with respect to η. One possible weakening of the definition is to allow the size of S to grow with η:

Definition 29 *The encryption is* non-uniformly ideal *if it satisfies:*

$$\begin{aligned}
&\exists,\ \textit{a polynomial } p\ \forall\, \mathtt{A},\ \forall\, \textit{polynomials } q,\ \forall\, s.\,l.\,\eta \\
&\forall S \subset \mathcal{A}(\textit{such that } |S| \leq p(\eta)),\ \forall M \notin \left(\mathcal{A} \setminus C\left[\widehat{S}\right]\right),\ \forall \pi \in \Pi_\eta \\
&\Pr[\quad f_\eta \leftarrow \mathsf{Keymaps}_\eta; \\
&\qquad \omega \leftarrow \Omega_\eta; \\
&\qquad s \leftarrow [\![S]\!]; \\
&\qquad m \leftarrow \mathtt{A}^{\mathit{PubKeyOf}(\cdot),\mathit{PrivKeyOf}(\cdot)}(1^\eta, s, t\,(S)) : \\
&\qquad m \in [\![M]\!] \qquad\qquad\qquad\qquad] \leq \tfrac{1}{q(\eta)}
\end{aligned}$$

In other areas, it would be desirable to extend this work to encompass other cryptographic operations, such as secret-key cryptography. It may be that the secret-key analog of ideal public-key cryptography has already been identified under some other name (such as "authenticated encryption"), but the connection to the Dolev–Yao adversary has not been explored.

Lastly, and most importantly, the results in this work contain only half of the picture. In particular, the results here show that if the adversary's objective is to produce a legitimate encoding at each query, then the trace of such queries and responses must correspond to one available to the Dolev–Yao adversary. But what if the adversary instead chooses to send an illegitimate encoding? Perhaps the adversary can reasonably hope to gain information about the plaintext of a given encryption by using honest participants to tell it what strings are valid encodings and which are not.

Similarly, what should a regular participant do if what it receives is not an encoding of the message the participant expects at that time? Should it send an error, assuming that the message was garbled in transit, and accept resubmissions? Should it send an error and terminate? Should it simply terminate? All three of these may seem like safe courses of action at first glance, but the question deserves deeper examination. In each case, the behavior of the regular participant indicates that an error occurred[1] which reveals a single bit of information about the message to the adversary. This may be sufficient to break the underlying cryptography; otherwise secure protocols and implementations have been successfully attacked through an exploitation of error conditions. For example, it has been demonstrated that descriptive error codes provide the adversary with enough information to launch a chosen-ciphertext attack against a widely-used form of RSA [11, 4].

Hence, the way in which honest participants handle errors may undermine the security provided by the cryptography. Future work must address this issue, and must do it by analyzing the behavior of the participants in depth. In particular, the environment, which we abstracted away in our paper, must be examined in detail. The regular participants are typically underspecified in protocol specifications, particularly in terms of error handling. The most important extension of this work would be to formulate a characterization of honest participant behavior that gives the adversary no more advantage than it would gain through quasi-formal attacks.

[1]Assuming that the termination in the third case can be recognized by the adversary, by e.g. using timers.

Bibliography

[1] Martín Abadi and Phillip Rogaway. Reconciling two views of cryptography (the computational soundness of formal encryption). In *IFIP International Conference on Theoretical Computer Science (IFIP TCS2000)*, Lecture Notes in Computer Science. Springer-Verlag, 2000.

[2] Mihir Bellare and Phillip Rogaway. Entity authentication and key distribution. In D. Stinson, editor, *Advances in Cryptology–Crypto 93 Proceedings*, volume 773 of *Lecture Notes in Computer Science*. Springer–Verlag, 1993. Full version of paper available at http://www-cse.ucsd.edu/users/mihir/.

[3] Mihir Bellare and Phillip Rogaway. Optimal asymmetric encryption– how to encrypt with RSA. In A. De Santis, editor, *Advances in Cryptology – Eurocrypt 94 Proceedings*, volume 950 of *Lecture Notes in Computer Science*. Springer-Verlag, 1995.

[4] Daniel Bleichenbacher. Chosen ciphertext attacks against protocols based on the RSA encryption standard PKCS #1. In Hugo Krawczyk, editor, *Advances in Cryptology — CRYPTO '98*, volume 1462 of *Lecture Notes in Computer Science*, 1998.

[5] Manual Blum, Alfredo De Santis, Silvio Micali, and Giuseppe Persaino. Noninteractive zero knowledge. *SIAM Journal on Computing*, 20(6):1084–1118, December 1991.

[6] D. Dolev and A. Yao. On the security of public-key protocols. *IEEE Transactions on Information Theory*, 29:198–208, 1983.

[7] Oded Goldreich. Founcations of cryptography. Early, unpublished version of manuscript, July 2000.

[8] Joshua Guttman, Javier Thayer Fábrega, and Lenore Zuck. The faithfulness of abstract encryption. Forthcoming, 2001.

[9] Gavin Lowe. An attack on the Needham–Schroeder public-key authentication protocol. *Information Processing Letters*, 56:131–133, 1995.

[10] Gavin Lowe. Breaking and fixing the Needham–Schroeder public-key protocol using FDR. In Margaria and Steffen, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, volume 1005 of *Lecture Notes in Computer Science*, pages 147–166. Springer Verlag, 1996.

[11] James Manger. A chosen ciphertext attack on RSA optimal asymmetric encryption paddin OAEP as standardized in PKCS #1 v2.0. In *Advances in Cryptology — CRYPTO 2001*, 2001.

[12] Silvio Micali, Charles Rackoff, and Bob Sloan. The notion of security for probabilistic cryptosystems. *SIAM Journal on Computing*, 17(2), April 1988.

[13] John C. Mitchell, Mark Mitchell, and Ulrich Stern. Automated analysis of cryptographic protocols using Murφ. In *Proceedings, 1997 IEEE Symposium on Security and Privacy*, pages 141–151. IEEE, Computer Society Press of the IEEE, 1997.

[14] Roger Needham and Michael Schroeder. Using encryption for authentication in large networks of computers. *Communications of the ACM*, 12(21):993–999, 1978.

[15] L C Paulson. The inductive approach to verifying cryptographic protocols. *Journal of Computer Security*, 6:85–128, 1996.

[16] Alfredo De Santis, Giovanni Di Crescenzo, Rafail Ostrovsky, Giuseppe Persiano, and Amit Sahai. Robust non-interactive zero knowledge. In *Advances in Cryptology–CRYPTO 2001 Proceedings*, 2001.

[17] F. Javier Thayer Fábrega, Jonathan C. Herzog, and Joshua D. Guttman. Strand spaces: Proving security protocols correct. *Journal of Computer Security*, 7(2/3):191–230, 1999.

Appendix A

Index of notation

A.1 Formal sets and operators

We use the following notation for sets and operations in the setting of formal cryptography:

$A \to B : M$	Entity A sends the message M, addressed to B. (No guarantee that it reaches B.)
$\{\|M\|\}_K$	Message M encrypted using key K.
$M\,N$	The concatenation of messages M and N
$\mathcal{A}$	The formal algebra of terms
$\mathcal{T}$	The set of plaintexts
$\mathcal{M}$	The set of names
$Subv$	The set of names of subverted principals
$\mathcal{R}$	The set of nonces
$\mathcal{R}_{Adv}$	Nonces whose values can be set by the adversary
$\mathcal{R} \setminus \mathcal{R}_{Adv}$	Nonces whose values are picked randomly by honest participants
$\mathcal{K}_{Pub}$	The set of public keys
$\mathcal{K}_{Priv}$	The set of private keys
$\mathcal{K}_{Adv}$	Keys whose values can be set by the adversary
$\mathcal{K}_{Subv}$	Keys of principals subverted by the adversary after their keys have been chosen
$\mathcal{K}_{Bad}$	All private keys known to the adversary ($= \mathcal{K}_{Adv} \cup \mathcal{K}_{Subv}$)
$\mathcal{K}$	The set of keys
$\mathcal{E}$	The set of encryptions
$\mathcal{C}$	The set of concatenations
K^{-1}	The private key associated with public key K
K_A	The public key of entity A
$C\,[S]$	The closure of set S
$pattern_{sk}\,(M)$	The symmetric-key pattern of M
$pattern_{pk}\,(M, T)$	The public-key pattern of M Given the set of keys T.
$\equiv$	Equivalence of expressions
$\cong$	Equivalence up to renaming
$t\,()$	Function from expressions to tags
$\widehat{S}$	The knowledge set over S ($= S \cup \mathcal{M} \cup \mathcal{K}_{Pub} \cup \mathcal{K}_{Subv} \cup \mathcal{R}_{Adv}$)
$S\|_M$	Elements of S which have M in their parse tree
$\overline{S}$	The concatenation of S; the message created by pairing together all the elements of S in some canonical order
$\mathcal{K}\,(M)$	$\mathcal{K}\|_M \cap (\mathcal{K}_{Pub} \cup \mathcal{K}_{Adv})$; the set of all keys potentially useful in the analysis of M

A.2 Computational sets and operations

We use the following notation for sets and operations in the setting of computational cryptography:

η	Security parameter
q	A polynomial, usually used to be a function of η
$D_\eta \cong D'_\eta$	Distribution families D_η and D'_η are computationally indistinguishable
N	The natural numbers
$x \leftarrow D$	x is drawn from the distribution D. (If D is a set, x is drawn uniformly from D.)
$\Pr[a : p]$	The probability that predicate p is true given a
G	The key generation algorithm (symmetric or asymmetric, depending on context)
k	A computational symmetric key
e	A computational encryption key
d	A computational decryption key
E	The (computational) encryption algorithm (symmetric or asymmetric, depending on context)
c	A computational ciphertext
D	The (computational) decryption algorithm (symmetric or asymmetric, depending on context)
P	Key extractor, which derives a public key from any encryption made with that key
$\mathsf{Parameter}$	The set of security parameters (typically N)
Coins	The set of infinite random bit sequences
$\mathsf{PublicKey}$	The set of possible public keys
$\mathsf{PrivateKey}$	The set of possible private keys
$\mathsf{Ciphertext}$	The set of possible encryptions
String	The set of finite bit strings
$\mathsf{E}_k(m)$	The probability distribution induced $\mathsf{E}(m, r, k)$ when r is chosen randomly (uniformly) from Coins.
Key	The set of possible secret (symmetric) keys
$\mathtt{A}$	The adversary
$\mathtt{P}$	The prover algorithm of a ZK proof
$\mathtt{V}$	The verifier algorithm of a ZK proof
$O(\cdot)$	The random oracle
x, p, w, σ	In the context of NIZK proofs, a theorem, a proof, a witness, and a reference string (respectively)
$b_{\langle m,n \rangle}$	The contiguous substring of b starting at position m and ending at position n (inclusive).
C	Conversion verifier

A.3 Sets and operations used to connect formal and computational settings

We use the following notation for sets and operations used to bridge and connect the formal and computational settings:

$[M]$	The distribution of M, induced by running the `Convert` algorithm on M
$\mathsf{Keymaps}_\eta$	Family (indexed by η) of distributions of functions from formal keys to computational keys
f	Random element of $\mathsf{Keymaps}_\eta$
μ	Function from formal name to bit-strings
Ω_η	Set of functions from $\mathcal{R} \setminus \mathcal{R}_{Adv}$ to bit-strings
ω	Randomly chosen element of Ω_η
Π_η	Set of functions from $\mathcal{R}_{Adv}$ to bit-strings
π	Element of Π_η (chosen by adversary)
Ψ_η	Set of functions from $\mathcal{K}_{Adv}$ to bit-strings
ψ	Element of Ψ_η (chosen by adversary)
$\mathtt{Init}_\eta^n[A, B]$	Algorithm corresponding to the Needham–Schroeder initiator, with A as initiator, B as responder, security parameter η, instance n.
$\mathtt{Resp}_\eta^n[A, B]$	Algorithm corresponding to the Needham–Schroeder responder, with A as responder, B as initiator, security parameter η, instance n.
`PubKeyOf`	Function from names to public keys
`PrivKeyOf`	Function from names (of subverted principals) to private keys

Appendix B

An Abadi–Rogaway-like result for public-key encryption

In this chapter, we give a proof for Theorem 16 using the `Convert` function given in Chapter 4:

Theorem 16 *Suppose M and N are acyclic expressions. If $pattern_{pk}\left(M,T\right) = pattern_{pk}\left(M,T\right)$ for some $T \subseteq \mathcal{K}$, then $[\![M]\!] \cong [\![N]\!]$.*

Proof: We prove this by hybrid argument. Since M and N are acyclic, we can order the keypairs used in the parse tree of M as p_{M_1}, $p_{M_2} \ldots p_{M_k}$ so that if $p_i \to p_j$ in the graph G_M, then $i \geq j$. That is, the deeper the key in the encryptions, the smaller the number. We will write $K^{-1}_{M_i}$ to mean the private key associated with p_{M_i}. The keys in N receive a similar notation.

We go about the hybrid argument by constructing a number of intermediate patterns between M and N. In particular, we construct patterns M_0, M_1,... M_k and N_0, N_1... N_l such that:

- $M_0 = M = pattern_{pk}\left(M, T \cup \left\{K^{-1}_{M_1}, K^{-1}_{M_2}, \ldots K^{-1}_{M_k}\right\}\right)$ and similarly for N_0,
- $M_k = N_l = pattern_{pk}\left(M,T\right) = pattern_{pk}\left(N,T\right)$, and
- $M_i = pattern_{pk}\left(M, T \cup \left\{K^{-1}_{M_{i-1}}, K^{-1}_{M_{i-2}}, \ldots K^{-1}_{M_k}\right\}\right)$ and similarly for N_i.

That is, between N_i and N_{i+1} we pick a key K, and replace all encryptions with that key with $\Box_K$. For example, suppose

$$M = \{\!|A|\!\}_{K_1} \; \left\{\!\left|K_1^{-1}\right|\!\right\}_{K_2} \; \{\!|B|\!\}_{K_3} \; \{\!|A\,B|\!\}_{K_2}\,,$$

$$N = \{\!|K_2^{-1}|\!\}_{K_1}\ \{\!|C\,D|\!\}_{K_2}\ \{\!|B|\!\}_{K_3}\ \{\!|A|\!\}_{K_2}\,,$$

and

$$T = \left\{K_3^{-1}\right\}$$

Both of these have the pattern, given T, of:

$$\mathit{pattern}_{pk}\,(M,T) = \mathit{pattern}_{pk}\,(N,T) = \Box_{K_1}\,\Box_{K_2}\ \{\!|B|\!\}_{K_3}\ \Box_{K_2}$$

By using the order on keys suggested by the notation, we can let

$$\begin{array}{rcllll}
M_0 = M & = & \{\!|A|\!\}_{K_1} & \{\!|K_1^{-1}|\!\}_{K_2} & \{\!|B|\!\}_{K_3} & \{\!|A\,B|\!\}_{K_2} \\
M_1 & = & \Box_{K_1} & \{\!|K_1^{-1}|\!\}_{K_2} & \{\!|B|\!\}_{K_3} & \{\!|A\,B|\!\}_{K_2} \\
M_2 & = & \Box_{K_1} & \Box_{K_2} & \{\!|B|\!\}_{K_3} & \Box_{K_2} \\
M_3 = N_3 & = & \Box_{K_1} & \Box_{K_2} & \{\!|B|\!\}_{K_3} & \Box_{K_2} \\
N_2 & = & \Box_{K_1} & \Box_{K_2} & \{\!|B|\!\}_{K_3} & \Box_{K_2} \\
N_1 & = & \{\!|K_2^{-1}|\!\}_{K_1} & \Box_{K_2} & \{\!|B|\!\}_{K_3} & \Box_{K_2} \\
N_0 = N & = & \{\!|K_2^{-1}|\!\}_{K_1} & \{\!|C\,D|\!\}_{K_2} & \{\!|B|\!\}_{K_3} & \{\!|A|\!\}_{K_2}
\end{array}$$

For each of these patterns, we can associate a probability distribution on bit-strings. The probability distributions $[\![M]\!]$ and $[\![N]\!]$ are well-defined, and we can define a probability distribution for each of the other patterns in the following way: let $[\![\Box]\!]$ be a fixed bit-string (which can depend on η). Then we can define $[\![\Box_K]\!]$ to be the encryption of $[\![\Box]\!]$ under the appropriate computational key:

$$\langle \mathcal{E}\left([\![\Box]\!], U\left[\mathsf{Coins}\right], f_\eta(K)\right), \text{“enc”}\rangle$$

With this, we can define the distribution associated with a pattern recursively, as in Chapter 4.

Now, suppose that the distributions $[\![M]\!]$ and $[\![N]\!]$, the top and bottoms rows of our table, are distinguishable. Then we know by hybrid argument that two consecutive rows are distinguishable. (The number of rows in the table is constant with respect to η.) Then we can continue the hybrid argument by creating a new table between the two distinguishable rows. Suppose that K_i is the key being “blobbed” between the two rows. Then there are a fixed number of encryptions being converted two or from “blobs”. For example, if the two rows are

$$M_1 = \Box_{K_1}\ \{\!|K_1^{-1}|\!\}_{K_2}\ \{\!|B|\!\}_{K_3}\ \{\!|A\,B|\!\}_{K_2}$$

and

$$M_2 = \Box_{K_1}\, \Box_{K_2}\, \{\!|B|\!\}_{K_3}\, \Box_{K_2}$$

Then we could expand this into the table:

$$\begin{array}{rcllll} M_1 & = & \Box_{K_1} & \{\!|K_1^{-1}|\!\}_{K_2} & \{\!|B|\!\}_{K_3} & \{\!|A\,B|\!\}_{K_2} \\ M_{1.5} & = & \Box_{K_1} & \Box_{K_2} & \{\!|B|\!\}_{K_3} & \{\!|A\,B|\!\}_{K_2} \\ M_2 & = & \Box_{K_1} & \Box_{K_2} & \{\!|B|\!\}_{K_3} & \Box_{K_2} \end{array}$$

Again, the number of rows is constant with respect to η, so there must exist two consecutive rows that can be distinguished. Assume, without loss of generality, that it is rows M_1 and $M_{1.5}$. Hence, we can use these two distributions to build an adversary that can distinguish between, e.g. $[\![\{\!|K_1^{-1}|\!\}_{K_2}]\!]$ and $[\![\Box_{K_2}]\!]$, which will allow us to break the semantic security of the computational encryption scheme. Note that in general, we will distinguish between $[\![\Box_K]\!]$, the encryption of the fixed message $[\![\Box]\!]$, and $[\![\{\!|p|\!\}_K]\!]$ for some fixed pattern p. Because of the way the first hybrid table was constructed we know that the pattern p will not contain any encryptions, but may contain one or more instances of $\Box_K$. Hence, we are trying to distinguish between the encryption of a fixed message on the one hand and the encryption of a random variable on the other. However, the definition of semantic security (Definition 5) allows for this generality. ■

As a corollary, we note that every message is indistinguishable from its pattern:

Theorem 17 *For any $T \subseteq \mathcal{K}$, $[\![M]\!] \cong [\![pattern_{pk}(M,T)]\!]$.*

One can see this in two different ways. First, we note that if we extend Definition 14 to include

- $p'(\Box_K, T) = \begin{cases} \Box_K & \text{if } K^{-1} \notin T \\ \text{undefined} & \text{otherwise} \end{cases}$

then

$$pattern_{pk}\left(pattern_{pk}\left(M,T\right),T\right) = pattern_{pk}\left(M,T\right).$$

An alternate proof is that as part of the proof for Theorem 16 above we showed that any two rows in the first table are indistinguishable. The message M and $pattern_{pk}(M,T)$ are both rows in the table. Hence, any message is indistinguishable from its pattern under any set of keys

Appendix C

Semantic Security of $(\mathsf{G}, \mathsf{E}, \mathsf{D})$

Recall the definition of $(\mathsf{G}, \mathsf{E}, \mathsf{D})$ from Theorem 26:

- $\mathsf{G} = \mathsf{G}'$
- $\mathsf{E}_e(m)$ be the distribution:

$$c \leftarrow \mathsf{E}'_c(m);\ p \leftarrow \mathtt{P}(c, \langle m, r \rangle, O(c)_{\langle 1, l(\eta) \rangle}) : \langle c, p \rangle$$

(where r is the randomness used by $\mathsf{E}'_e(m)$ to produce c), and

- $\mathsf{D}(\langle c, p \rangle, e) = \begin{cases} m & \text{if } \mathsf{D}(c, d) = m \text{ and } \mathtt{V}(c, p, O(c)_{\langle 1, l(\eta) \rangle}) = 1 \\ \perp & \text{otherwise} \end{cases}$

where $(\mathsf{G}', \mathsf{E}', \mathsf{D}')$ is a semantically secure encryption scheme, $O(\cdot)$ is the random oracle, and $(l, \mathtt{P}, \mathtt{V}, \mathtt{S}, \mathtt{E})$ is a reference-string specific, non-interactive zero-knowledge proof of knowledge system for $\mathsf{Ciphertext}$.

Recall also the definition of one-pass semantic security (Definition 5): $(\mathsf{G}, \mathsf{E}, \mathsf{D})$ is one-pass semantically secure if:

$$\forall \text{ PPT algorithms } \mathtt{M} \text{ and } \mathtt{A}, \forall \text{ polynomials } q, \forall \text{ s.l. } \eta,$$

$$\begin{aligned} \Pr[\quad & (e, d) \leftarrow \mathsf{G}(1^\eta); \\ & m_0, m_1 \leftarrow \mathtt{M}(1^\eta, e); \\ & b \leftarrow \{0, 1\}; \\ & \langle c, p \rangle \leftarrow \mathsf{E}_e(m_b) : \\ & \mathtt{A}(1^\eta, e, m_0, m_1, \langle c, p \rangle) = b \quad] \leq \tfrac{1}{2} + \tfrac{1}{q(\eta)} \end{aligned}$$

(We use notation $\forall$ s.l. η to mean "for all sufficiently large η.") So, to prove that $(\mathsf{G}, \mathsf{E}, \mathsf{D})$ is semantically secure, let us suppose that it is not. That is, assume that:

$$\exists \text{ PPT algorithms } \mathtt{M} \text{ and } \mathtt{A}, \exists \text{ a polynomial } q, \text{ for infinitely many } \eta,$$

$$\begin{aligned}\Pr[\quad & (e,d) \leftarrow \mathsf{G}(1^\eta);\\ & m_0, m_1 \leftarrow \mathtt{M}(1^\eta, e);\\ & b \leftarrow \{0,1\};\\ & \langle c,p\rangle \leftarrow \mathsf{E}_e(m_b) :\\ & \mathtt{A}(1^\eta, e, m_0, m_1, \langle c,p\rangle) = b \quad] \geq \tfrac{1}{2} + \tfrac{1}{q(\eta)}\end{aligned}$$

From this, we will build algorithms $\mathtt{M}'$ and $\mathtt{A}'$ that defeat the semantic security of $(\mathsf{G}', \mathsf{E}', \mathsf{D}')$:

- Let $\mathtt{M}' = \mathtt{M}$.
- The algorithm $\mathtt{A}'$ will use both $\mathtt{A}$ and $\mathtt{S} = (\mathtt{S}_1, \mathtt{S}_2)$, the simulator for the NIZK proof system. In particular:

  $\mathtt{A}'$ = on input $(1^\eta, e, m_0, m_1, c)$
  - Runs $\mathtt{S}_1(1^\eta)$ to get (σ, τ),
  - Runs $\mathtt{S}_2(c, \sigma, \tau)$ to get p, and
  - Runs $\mathtt{A}(1^\eta, e, m_0, m_1, \langle c,p\rangle)$ to get b.
  - Lastly, $\mathtt{A}'$ returns b.

What is going on here? The new adversary $\mathtt{A}'$ uses the simulator $\mathtt{S}$ to produce a "proof" for e which is indistinguishable from a real proof. That is, by Definition 9, the behavior of $\mathtt{A}$ on $\langle c,p\rangle$ will be computationally indistinguishable from that where $\langle c\rangle$ is a theorem and $\langle p\rangle$ is given by the prover $\mathtt{P}$.[1] Hence, the input $(1^\eta, e, m_0, m_1, \langle c,p\rangle)$ is computationally indistinguishable from the input given to $\mathtt{A}$. Since $\mathtt{A}$ can distinguish between an encryption of m_0 and m_1 under the scheme $(\mathsf{G}, \mathsf{E}, \mathsf{D})$, it must be that the adversary $\mathtt{A}'$ can distinguish between an encryption of m_0 and m_1 under the scheme $(\mathsf{G}', \mathsf{E}', \mathsf{D}')$.

[1]To be formal, Definition 9 uses a bipartite adversary $\mathtt{A}_1$ and $\mathtt{A}_2$. Here, we assume that $\mathtt{A}_1(\sigma)$ returns $\langle c, \epsilon, (1^\eta, e, m_0, m_1)\rangle$ (where ϵ is the empty string) and $\mathtt{A}_2 = \mathtt{A}$.